AFRL-RI-RS-TR-2020-078

DEPENDABLE EXPERIMENTATION FOR SPACE/TIME ANALYSIS FOR CYBERSECURITY

APOGEE RESEARCH, LLC

MAY 2020

FINAL TECHNICAL REPORT

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

STINFO COPY

AIR FORCE RESEARCH LABORATORY
INFORMATION DIRECTORATE

■ **AIR FORCE MATERIEL COMMAND** ■ **UNITED STATES AIR FORCE** ■ **ROME, NY 13441**

NOTICE AND SIGNATURE PAGE

Using Government drawings, specifications, or other data included in this document for any purpose other than Government procurement does not in any way obligate the U.S. Government. The fact that the Government formulated or supplied the drawings, specifications, or other data does not license the holder or any other person or corporation; or convey any rights or permission to manufacture, use, or sell any patented invention that may relate to them.

This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09. This report is available to the general public, including foreign nations. Copies may be obtained from the Defense Technical Information Center (DTIC) (http://www.dtic.mil).

AFRL-RI-RS-TR-2020-078 HAS BEEN REVIEWED AND IS APPROVED FOR PUBLICATION IN ACCORDANCE WITH ASSIGNED DISTRIBUTION STATEMENT.

FOR THE CHIEF ENGINEER:

/ S /
WALTER S. KARAS
Work Unit Manager

/ S /
JAMES S. PERRETTA
Deputy Chief, Information
Exploitation & Operations Division
Information Directorate

This report is published in the interest of scientific and technical information exchange, and its publication does not constitute the Government's approval or disapproval of its ideas or findings.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. REPORT DATE *(DD-MM-YYYY)*	2. REPORT TYPE	3. DATES COVERED *(From - To)*
MAY 2020	FINAL TECHNICAL REPORT	MAY 2015 – DEC 2019

4. TITLE AND SUBTITLE
DEPENDABLE EXPERIMENTATION FOR SPACE/TIME ANALYSIS FOR CYBER SECURITY

5a. CONTRACT NUMBER
FA8750-15-C-0089

5b. GRANT NUMBER
N/A

5c. PROGRAM ELEMENT NUMBER
61101E

6. AUTHOR(S)
Evan Fortunato, Kwame Ampeh, Christina Wise,
Anders Jagd, Scott Laprise, and Greg Frazier

5d. PROJECT NUMBER
STAC

5e. TASK NUMBER
OG

5f. WORK UNIT NUMBER
EE

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
Apogee Research, LLC
4075 Wilson Blvd Suite 600
Arlington VA 22203

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
Air Force Research Laboratory/RIGA
525 Brooks Road
Rome NY 13441-4505

10. SPONSOR/MONITOR'S ACRONYM(S)
AFRL/RI

11. SPONSOR/MONITOR'S REPORT NUMBER
AFRL-RI-RS-TR-2020-078

12. DISTRIBUTION AVAILABILITY STATEMENT
Approved for Public Release; Distribution Unlimited. This report is the result of contracted fundamental research deemed exempt from public affairs security and policy review in accordance with SAF/AQR memorandum dated 10 Dec 08 and AFRL/CA policy clarification memorandum dated 16 Jan 09

13. SUPPLEMENTARY NOTES

14. ABSTRACT

The Space/Time Analysis for Cybersecurity (STAC) program set out to identify new program analysis techniques and create new tools to asses vulnerabilities related to the space and time resource usage behavior of algorithms. STAC sought to enable analysts to both identify and rule out these vulnerabilities in software at levels of scale and speed great enough to support a methodical search for them in the software upon which the U.S. government, military, and economy depend. The results of the final engagement show that under the assumption that the Control Team represents the state-of-the-art in the five vulnerability categories, the STAC R&D performers advanced the state-of-the-art in Algorithmic Complexity vulnerabilities in Time and Space, and in Side Channel vulnerabilities in Time.

15. SUBJECT TERMS
Space/Time Analysis, Side Channel Vulnerabilities, Algorithmic Complexity, Memory Resource Usage, Encryption Side Channel

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			**WALTER S. KARAS**
U	U	U	UU	1205	19b. TELEPHONE NUMBER (*Include area code*) **N/A**

TABLE OF CONTENTS

List of Tables

List of Figures

List of Equations

1 Summary

The four-year STAC program researched approaches to identify and rule out Algorithmic Complexity and Side Channel vulnerabilities in Time and Space. These algorithmic resource usage vulnerabilities are categorized into two classes: Side Channel vulnerabilities and Algorithmic Complexity vulnerabilities. Side Channel vulnerabilities are information leakage caused by algorithmic computations on secret values, while Algorithmic Complexity vulnerabilities are often the result of human programming errors that result in excessive resource usage. Subcategorizing by the dimensions of Time and Space, results in the following five categories of vulnerabilities for the STAC program:

1. Algorithmic Complexity in Time
2. Algorithmic Complexity in Space
3. Side Channel in Time
4. Side Channel in Space
5. Side Channel in Time and Space

The results of the final engagement, best summarized in Figure 1, show that the STAC R&D performers advanced the state-of-the-art in Algorithmic Complexity vulnerabilities in Time and Space, and in Side Channel vulnerabilities in Time.

Responses to individual engagement challenges are included in Appedix A. A further study of Side Channel in Emissions is included as Appendix B.

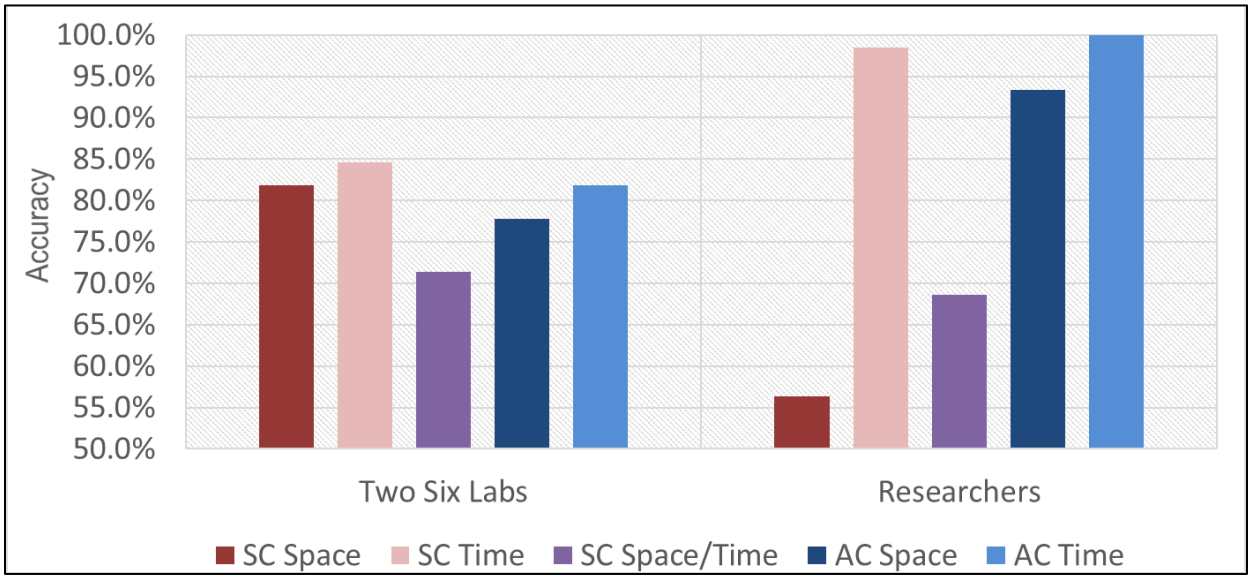


Figure 1: Engagement 7 accuracy by challenge question category, Control Team (Two Six Labs) versus R&D Performers

2 Introduction

The Space/Time Analysis for Cybersecurity (STAC) program set out to identify new program analysis techniques and create new tools to asses vulnerabilities related to the space and time resource usage behavior of algorithms. STAC sought to enable analysts to both identify and rule out these vulnerabilities in software at levels of scale and speed great enough to support a methodical search for them in the software upon which the U.S. government, military, and economy depend.

The STAC program experimental approach earlier in the program was focused on identifying which of the eight TA1 R&D performers had the strongest approach to identify and rule out vulnerabilities in the five categories. Following the fourth of the seven planned engagement events in the STAC program, it became clear that while different R&D performers had approaches and tools that handled part of the problem space, no R&D performer had an approach that covered the entire problem space. In response, the experimental approach shifted to focus on assessing the combined capabilities of the R&D performers. This change in experimental focus coupled with an analysis of the capabilities of different performers led to the identification of the five performers with non-overlapping approaches that provided the span of the capabilities of the R&D performers. The final STAC engagement fixed the total engineering effort for the participating R&D performers and the control team, and allowed us to ask the question of in which categories did the STAC R&D performers advance the state-of-the-art. The results of the final engagement, best summarized in Figure 2, show that under the assumption that the Control Team (Two Six Labs) represents the state-of-the-art in the five vulnerability categories, the STAC R&D performers advanced the state-of-the-art in Algorithmic Complexity vulnerabilities in Time and Space, and in Side Channel vulnerabilities in Time.

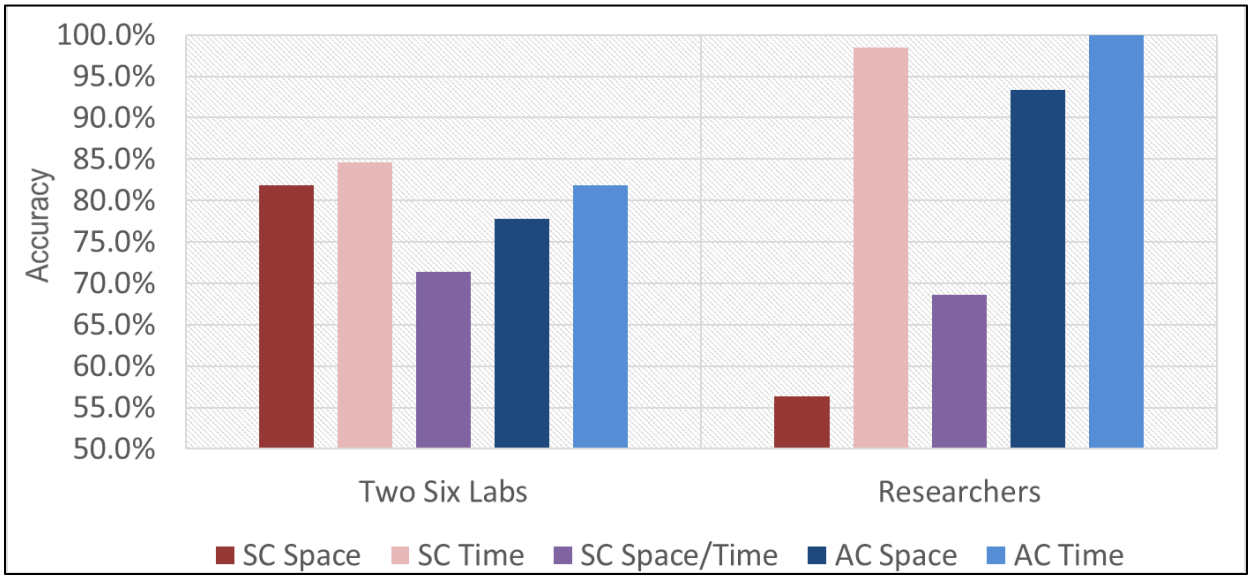


Figure 2: Engagement 7 accuracy by challenge question category, Control Team (Two Six Labs) versus R&D Performers

The remainder of this introduction discusses the experimental approaches to asses the performance of the R&D performers and the Control Team. The rest of the final report then looks at the experimental plan for each engagement and provides a thorough assessment of the results of each engagement. The conclusions section provides final concluding thoughts on the program, with more detailed experimental analysis of individual responses in Appendix A and a study of Side Channel in Emission as Appendix B.

2.1 Establishing Operational Definitions

At the beginning of the program, the goal of Apogee, as the experimental lead, was to identify an experimental construct to evaluate the capabilities of individual R&D performers (collectively known as the Blue Teams) relative to the Control Team. During each of the seven planned engagements, the Blue Teams would be provided a set of challenge questions each corresponding to a challenge program written by the Adversarial Challenge performers. The Blue Teams would have to analyze and respond whether a given challenge program was vulnerable or not vulnerable to the vulnerability category and scenario provided in each challenge question. Each R&D performer would then be assessed on their ability to both identify and rule out vulnerabilities relative to the Control Team. This experimental approach required clear and unambiguous definitions of vulnerability for each of the five vulnerability categories to answer the question of at what point is an application vulnerable. The concept of strength of vulnerability was borne out of this need for an unambiguous definition of vulnerability for the STAC program. Programs in general use time and memory resources, and most algorithms that reasons over secret data leak some information about the secret; however, an application is only declared vulnerable when it exceeds the thresholds for resource consumption or secret leakage established in the challenge question for that application. The operation definitions document (Appendix 4.4) and the engagement plans provided the unambiguous definitions of vulnerability used to assess the performers. Throughout the STAC program, these definitions were adapted to address ambiguities as they were identified.

2.2 Assessing Individual R&D Capabilities E1 – E4

The first four STAC engagements were a combination of live and take-home engagements designed to asses the individual capabilities of each performer's approach. Engagements 1 and 3 were live engagements with engagements 2 and 4 as the take-home engagements. The live engagements were intended to push the performers to make their tools usable, while the multi-month take-home engagements were designed to allow the performers to refine their approaches and improvements as they discovered cases that their current tools or approaches failed to account for. To further assist performers to identify invalid assumptions or corner cases in their approaches, Apogee developed a set of small example problems that addressed a potential weakness for a set of performers. These Apogee derived examples comprised issues identified from site visits with the performers, analysis of their engagement performances, and assumptions stated in their PI meeting presentations. Following the first four engagements, teams with lightweight tools excelled in the live engagements but failed to improve during take-home engagements. Teams with more time and resource-intensive tools performed much better in the take-home format, but struggled at the faster-paced live engagements. In these first four engagements, the performers were not required to answer all challenge questions, but were instead allowed to select only those questions where their tools performed best. The accuracy of the R&D performers and the Control Team over all the challenge questions in Engagement 4,

Figure 3, shows that no single performer was able to achieve a high degree of accuracy across all five vulnerability categories.

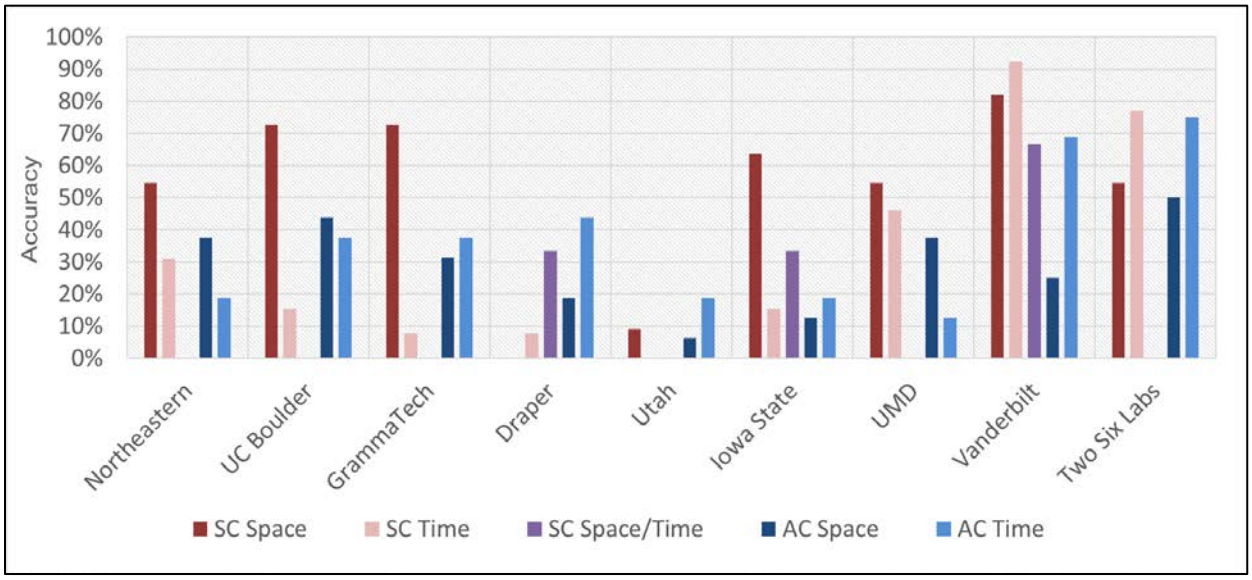


Figure 3: Engagement 4 accuracy by challenge question category

2.3 Assessing Combined R&D Capabilities E5 – E7

The experimental approach for Engagement 5 onwards changed to focus on assessing the combined capabilities of the R&D performers. This change was coupled with a change to a more realistic experimental platform with the server and client running on different machines communicating via a network switch rather than via localhost on a single machine as with Engagements 1 through 4. Though this change was motivated to increase realism, it also removed the timing variations introduced by running both client and server on the same machine and separated the attacker onto a different machine, further clarifying what information the attacker has access to. Engagements 5 and 6 contained both a take-home and live collaborative component, where all performers (including the control team) worked collaboratively on the challenge questions. This approach helped set the stage for Engagement 7 – where the five R&D performers, with the most distinguished approaches that best spanned the space of STAC vulnerabilities, worked collaboratively to assess the capabilities of the STAC R&D performers as a collective when compared to the current state-of-the-art.

3 Methods, Assumptions, and Procedures

3.1 Operational Definitions

3.1.1 Introduction

This document contains guidance for all teams regarding the upcoming engagement. It provides an operational definition of vulnerability to algorithmic complexity attack and an operational definition of side channel vulnerability. AC Teams can use these operational definitions to determine when the vulnerabilities they've inserted into their challenge programs are severe enough for use in engagements. The Control and R&D Teams can use these operational definitions to determine when they've found a vulnerability worth reporting. There is also some guidance on scope and on the information that will pass between the teams during engagements. This guidance should be good enough to guide you're planning for the time being; DARPA and the EL Team may send out updated guidance before the engagement.

Many thanks to our EL and AC Teams for their help in forming these operational definitions! This document describes only a small part of the reasoning that led us to these definitions. The EL and AC Teams should feel free to share the supporting material they produced with the other STAC performers as they see fit.

3.1.2 Operational Definition: Vulnerability to Algorithmic Complexity Attack

In STAC engagements, R&D Teams will identify which challenge programs contain vulnerabilities to algorithmic complexity attack and which do not. For the purposes of STAC engagements, a challenge program contains a vulnerability to algorithmic complexity attack if and only if it is possible for an adversary to cause that challenge program to exceed a specific resource usage limit with a defined probability of success after feeding it some number of bytes of input, where that number of bytes is less than a specific input budget. Note, the adversary should be able to cause that challenge program to exceed the specified resource usage limit (equaling or exceeding the defined probability of success) for any benign background interaction. In cases where the resource is time, the following definitions will be used to distinguish between self-DoS and remote-DoS vulnerabilities.

3.1.2.1 Self-DoS Algorithmic Complexity in Time (AC Time) Vulnerabilities

A stand-alone challenge program contains an AC Time vulnerability if and only if a user can cause the application's response time to exceed the resource usage limit with a defined probability of success while remaining within the input budget.

3.1.2.2 Remote-DoS AC Time Vulnerabilities

A remote challenge program (server or peer-to-peer) contains an AC Time vulnerability if and only if it is possible for an adversary to cause a benign user's request to exceed the resource usage limit with a defined probability of success while remaining within the input budget. Resource usage will be evaluated for a reference benign user request (specified in the challenge question) rather than for any possible benign user interaction.

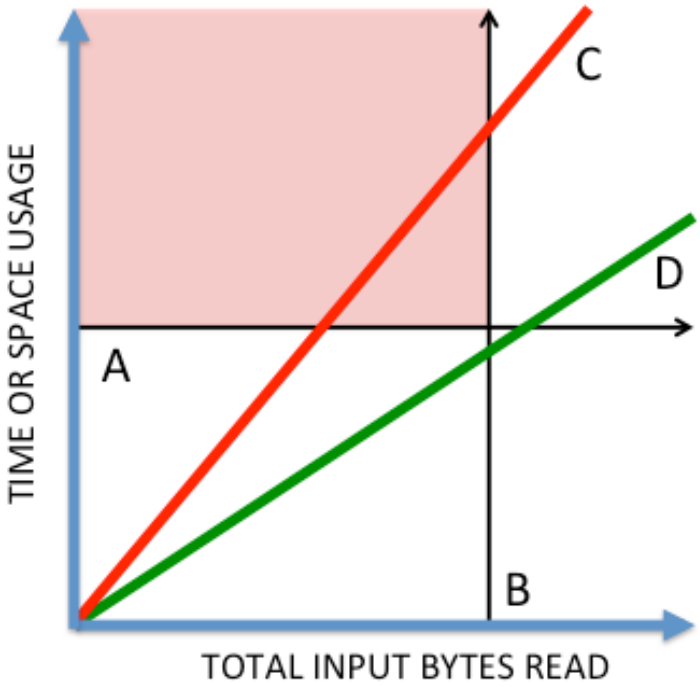


Figure 4: Idealized Graph of Resource Usage

Consider the graph in Figure 4. This graph is an idealized graph of the resource usage of a challenge program. The vertical axis describes resource usage. Depending on the challenge program, this axis might measure bytes of RAM, output file size in bytes, output packet payloads in bytes, program runtime in terms of Java Virtual Machine instructions executed, or some other relevant quantity. The horizontal axis describes the total number of bytes given to the challenge program as input.

Line A is the resource usage limit. Line B is the input budget. The shaded region indicates the presence of a vulnerability. A challenge program whose resource usage is bounded by line C contains a vulnerability. One whose resource usage is bounded by line D does not.

3.1.2.3 Scope: Internal vs. External Vulnerabilities

Some vulnerabilities to algorithmic complexity attack stem from the use of an algorithm that has two modes, one with low resource usage and the other with high resource usage. The programmer may have believed that the inputs that trigger the high resource usage mode would be rare, but an adversary finds a way to make those inputs common. Where scope is concerned, the key aspect of these algorithms is the branch that leads to the paths implementing the two different modes. If this branch occurs inside the challenge program, the vulnerability is in scope. If this branch occurs outside of the challenge program, the vulnerability is out of scope. R&D Teams should ignore out-of-scope vulnerabilities.

Examples of out-of-scope vulnerabilities involve triggering the slow modes of the operating system kernel's filesystem cache or memory paging mechanism, or the slow mode of the CPU's instruction or data caches. These may be valid real-world vulnerabilities, but they are out of

scope for STAC as we are limiting ourselves to the analysis of Java bytecode for programmatic reasons.

Note that it is the location of the branch that is important, not the location of the slowness. A vulnerability based on a challenge program with a branch that decided between a mode that writes RAM (fast) or removable media (slow) would be in scope. If the challenge program had no branch and instead called some abstract write API function that left the RAM/media decision up to the kernel's virtual filesystem, the vulnerability would be out of scope.

There is one exception to this inside/outside rule: vulnerabilities related to the behavior of the Java garbage collector are in scope, even though the branches in question are outside of the challenge program. We may decide to avoid vulnerabilities related to the behavior of the Java garbage collector in the first few engagements. See the Engagement-specific Limitations on Scope section below.

3.1.2.4 Scope: Environmental Constraints on Input

Some real-world programs would appear to have vulnerabilities to algorithmic complexity attack according to our operational definition. However, in a real-world deployment their vulnerabilities cannot manifest because the environment in which those programs run limits space of the possible inputs. For example, a network intrusion detection system might appear vulnerable, but the number of source IPv4 addresses the system might see during an actual deployment might be far smaller than all possible four-byte values due to the constraints of routing, and this smaller number might be insufficient to trigger the vulnerability.

Constraints imposed by the external environment that render an otherwise vulnerable program not vulnerable are out of scope for STAC. R&D Teams should follow the operational definition and declare these programs vulnerable rather than attempting to gather the application-domain-specific knowledge required to recognize that the vulnerability cannot occur in practice.

Note that challenge programs that attempt to filter input with explicit guard expressions in their code are in scope for STAC.

3.1.3 Operational Definition: Side Channel Vulnerability

In STAC engagements, R&D Teams will identify which challenge programs contain side channel vulnerabilities and which do not.

For the purposes of this definition, challenge programs take input, consume time and space resources, and contain a secret value of interest. Challenge programs take input in chunks that are domain-specific in nature: for example, a query, a request, or some other notion that may be different for each challenge program.

For the purposes of STAC engagements, a challenge program contains a side channel vulnerability if and only if an adversary can extract a worst-case complete secret value from the challenge program with a defined probability of success by executing a specifically bounded number of operations. This specific bound on the number of operations is called the adversary's operation budget.

Operations include:

> (1) Provide the challenge program with one input and observe the challenge program's outputs.

(2) Passively observe the challenge program's outputs without providing any input. DARPA will provide a definition of how much passive observation counts as one adversary operation. This definition may be different for each challenge program.

(3) Query a notional oracle with a guess.

This operational definition potentially excludes many valid real-world side channels that enable the adversary to deduce the secret with more operations than a particular operation budget allows. However, DARPA believes that techniques that are useful for reasoning about side channels encompassed by this limited definition will likely also be applicable to a broader notion of real-world side channels.

3.1.3.1 Worst-case Complete Secret Values

The notion of worst-case complete secret value deserves further explanation: Consider a password-checking program that supports passwords that are from one to eight characters in length where each character is drawn from the alphabet A to Z. Assume this program compares input guesses with the correct password value one character at a time working left to right. Instead of always making the full eight comparisons, it rejects bad guesses as soon as the first character fails to match and thus has the canonical textbook timing side channel.

An adversary might use this algorithm to exploit the side channel: guess values of the first character starting with A and running to Z. Once found, begin guessing the second character A to Z, and so on until the complete password value is found. With this adversary algorithm, the password "A" is the best-case complete secret value – it's the first password the algorithm with try and thus will likely be found in the least number of guesses. The password "ZZZZZZZZ" is the worst-case complete secret value – the algorithm will guess it last and thus it will likely take the most number of guesses to find. "Complete" indicates the value includes all characters of the password from beginning to end, as opposed to partial values that may match only some prefix of the complete password value. We define the adversary's operations budget in terms of this worst-case complete secret value. That is, a side channel meets the definition only if an adversary can extract the worst-case complete secret value within the operations budget – not the best-case complete secret value or some notion of average secret value.

3.1.3.2 Adversary Algorithms

The worst-case complete secret value depends on the algorithm the adversary uses to exploit the side channel. Different adversary algorithms may have different worst case complete secret values. For example, if the adversary algorithm in our previous example searched Z to A instead of A to Z, then its worst-case complete secret value would have been "AAAAAAAA" instead of "ZZZZZZZZ". Furthermore, the operations budget and ultimate probability of success may also depend on the algorithm the adversary uses. Sophisticated adversary algorithms may require fewer operations to extract secret values than simpler ones, and may do so with a higher probability of success. In our password side channel example, effective adversary algorithms would likely have to repeat each guess multiple times to overcome variance in its timing measurements. Simple algorithms might make a large fixed number of guesses for each character A to Z, choose the character that seems most likely to be correct, and move right to consider the next position. More complex algorithms might reduce the operations needed by optimistically moving right after only a few promising measurements and backtracking leftwards upon reaching a dead end.

In practice, Adversarial Challenge Teams will have to choose an adversary algorithm to use as the basis of their operations budget computation. Their choice may or may not be the most optimal algorithm. This implicit dependence on adversary algorithms is a weakness in our operational definition: DARPA cannot reveal its adversary algorithms to the R&D and Control Teams without simultaneously revealing the side channels hidden in challenge programs. But without revealing the adversary algorithms, there is no guarantee that the R&D and Control Teams will compute bounds matching DARPA's.

Presenting the R&D and Control Teams with an operations budget based on a particularly complex adversary algorithm is risky. For example, an R&D Team might produce a very precise picture of a challenge program's resource usage – a good result for STAC. However, if they base their side channel computations on a straightforward adversary algorithm that is significantly less optimal than the one DARPA used, they might miss a vulnerability and score poorly. This poor score would be a poor result for STAC since program analysis is more important to the research than thinking of clever adversary algorithms.

DARPA will work with the Adversarial Challenge and Experimentation Lead Teams to mitigate this risk, likely by inflating of operations budgets beyond what is strictly required by optimal but obscure adversary algorithms. This inflation will allow the R&D and Control Teams to confirm vulnerabilities with more obvious but sub-optimal adversary algorithms and ensure that scoring emphasizes program analysis.

3.1.3.3 Oracle Guesses

This definition uses the concept of oracle guesses to describe the amount of uncertainty left unresolved by observations. An oracle is an imaginary agent that accepts guesses and returns a yes/no response indicating whether or not those guesses are true. Guesses are proposed values for the complete secret. DARPA does not anticipate providing a separate piece of software that implements an oracle. However, at least some challenge programs may contain features that effectively implement this function. For example, a correctly implemented login program effectively provides an oracle by accepting complete passwords (the secret) and permitting or denying login (the indication of correctness).

3.1.3.4 Example

Here is an example of how to apply the above operational definition to a challenge program.

Imagine a challenge program that handles queries about health conditions over the network. The description of the challenge program says that information about the health conditions of patients shouldn't be revealed to anyone other than the service implemented by the challenge program and the individual patients themselves. Through some form of analysis the adversary determines that there are only 4 possible health conditions – a two-bit secret. They also determine that the size of the challenge program's network responses to queries may indicate what health condition the user has. This could be a side channel if it meets the operational definition. For this example, assume the adversary must achieve a 100% probability of success.

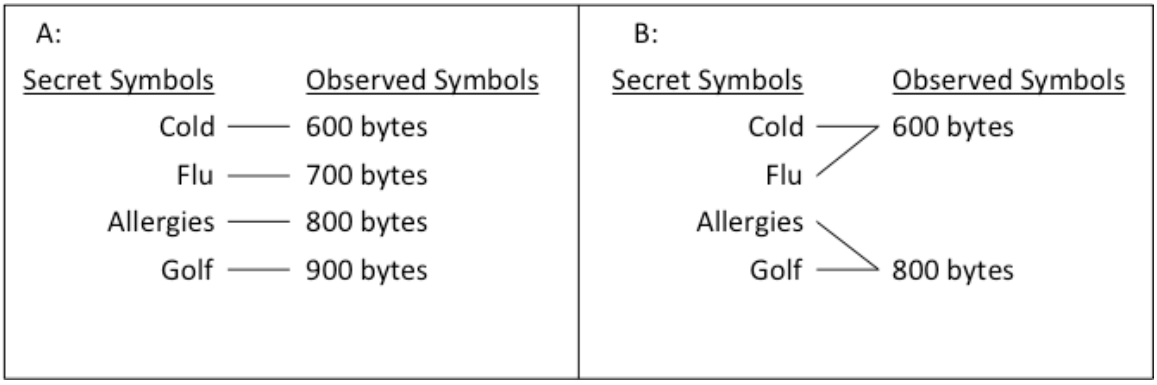


Figure 5: Mapping of secret to observed symbols

The analysis from here is best described in terms of secret symbols and observable symbols. There are 4 secret symbols, one for each health condition. Let's say there are four possible response sizes – that is, four possible observable symbols – and each health condition corresponds to exactly one size (Figure 5: Mapping of secret to observed symbols, Section A). In this case, the challenge program is directly revealing the health condition and the adversary can always determine the condition in one observation of an input-output operation. If the adversary's operation budget is at least 1, then this counts as a side channel.

Consider the more difficult case where there are only two possible response sizes (Figure 5: Mapping of secret to observed symbols, Section B). In this case, the adversary would need one observation of an input-output operation to narrow the possibilities to 2, and then one Oracle guess to figure out which of the two is correct. If the adversary's operation budget is 1, this case would not be a side channel. If the adversary's operation budget is 2 or more, it would be a side channel.

In practice, there can be further complexity: Perhaps there is noise that causes an observed symbol to map to more than one secret symbol, each with a particular probability. Perhaps the mapping changes with each observation. In this case, a side-channel is present if the secret can be obtained with a bounded probability of success acting within the adversary's operation budget.

3.1.4 Engagement-Specific Limitations on Scope

The previous sections of this document describe the scope DARPA intends to cover over the course of the STAC program. This section presents some temporary limitations that describe a narrower scope that we will use at least for the first engagement.

3.1.4.1 Common Reference Platform

Although STAC focuses on the analysis of Java programs, reasoning about these programs will require some understanding of the resource usage of operations occurring in the Java runtime, operating system, and underlying hardware. The amount of resources used by these operations may vary between different hardware/software platforms, particularly where time is concerned. This variation makes it difficult for the AC Teams to describe resource usage limits in a platform independent way. It also makes it difficult for them to produce side channel vulnerabilities that respect the same operation budget on all platforms.

To mitigate this difficulty, the AC Teams will describe the vulnerabilities they produce in terms of a common reference platform. The Control and R&D Teams are not required to perform their analyses on this common reference platform. However, DARPA will score their results in terms of the vulnerabilities that exist when the challenge programs run on this common reference platform. If the Control and R&D Teams choose different platforms they must handle whatever resource usage limit and operation budget translation is needed to make their results relevant to the common platform.

This is the common reference platform:

Intel Next Unit of Computing: NUC5i5RYH with SSD Samsung 850 EVO M.2 Sata 6Gb/s (Model # mz-n5e250bw) and Memory Crucial 16GB Kit 2-8GB PC3-12800 DDR3 KIT (Model # ct2kit102464BF160B).

Java run with just-in-time compilation (JIT) enabled.

Further details on the reference platform can be found in the reference platform document, *STAC-EL Reference Platform v3.2.3*.

Note: in cases where a client or peer application is available, unless otherwise specified in the description or the challenge question, the attacker can modify their version of the client or peer application to communicate with the server application or peer application under attack.

3.1.4.2 Encryption Side Channels

Brute force cracking of cryptographic keys is out of scope for STAC. The restrictions below will serve a final arbitration for whether a particular vulnerability/exploit is cracking cryptographic keys. For side channel vulnerabilities, Blue Teams can perform offline computations to model the system with the following limitations:

Time Limit: The post-processing of data (after the first bit of observed data is collected) is limited to 1 day of computing on the common reference platform.

Space Limit: Blue teams can pre-compute data for use in extracting a secret if the pre-computed data fits on the common reference platform.

3.1.4.3 Kinds of Observable Resource Usage

The AC Team's example vulnerabilities will involve only the kinds of space and time resource usage shown in Table 1: Vulnerabilities and Resources indicates which kinds of resource usage are relevant to which kinds of vulnerability.

Table 1: Vulnerabilities and Resources

ALG. COMP.	SIDE CHAN-NELS	
		TIME
X	X	The duration between pairs of the following kinds of events measured in wall-clock time: • Program start/termination. • Console input/output. • Filesystem input/output. • Network input/output.
		SPACE
X		Total space used by the Java Stack and Heap
X	X	Logical file size (as opposed to size on disk).
X	X	Size of Internet Protocol packet payload assuming adversary cannot trigger resends.

The EL Team will inform the Control and R&D Teams what instrumentation the AC Teams used to take their measurements.

3.1.4.4 Garbage Collection

In Engagements 5, the AC Teams will not produce any challenge programs with vulnerabilities that require a deep understanding of the Java garbage collector to analyze.

3.1.5 What Teams Will Receive and Produce

Here is a brief description of what the R&D, AC, and Control Teams will receive and produce for planning purposes. Expect an update from the EL Team as the engagement nears.

3.1.5.1 What AC Teams Will Deliver to DARPA

For each challenge program the AC Teams will deliver source and binary, plus:

> (1) A brief description of the challenge program's benign purpose. This description will include what kinds of information the challenge program should keep secret in terms of user-level abstractions (for example, passwords, patient health conditions, ID numbers). It will not indicate what data structures actually represent this information in the challenge program, nor will it indicate what secondary secrets should also be kept (for example, session keys).
>
> (2A) For challenge programs with vulnerabilities to algorithmic complexity attacks: resource usage limits and input budgets.
>
> (2B) For challenge programs with side channel vulnerabilities: operation budgets, and a target probability of success for extracting the complete worst-case secret.

(3) Some example inputs.

(4) A list of interfaces from which the challenge program might read input that is controlled by the adversary.

In a real-world deployment, R&D and Control Teams might begin the task of reasoning about vulnerabilities to algorithmic complexity attack by identifying all of the interfaces through which a challenge program reads input and using the details of how the challenge program is deployed to determine which interfaces might carry input controlled by an adversary. They might then concentrate their attention on the interfaces carrying adversary-controlled input as potential sources of attack, and ignore the others.

Although the AC Teams will strive to produce challenge programs that are as realistic as possible, the description in #1 may not always make it absolutely clear which interfaces might carry input controlled by the adversary. To avoid confusion, the AC Teams will deliver a list of interfaces through which the challenge program might read input controlled by the adversary. All interfaces will have an input budget. Some interfaces may have more than one budget to put limits on more than one kind of resource. Some interfaces may share budgets; some may have separate budgets. The AC Teams will deliver an explanation of how budgets relate to interfaces.

Note that the fact that an interface is on this list does not guarantee that the challenge program contains a vulnerability to algorithmic complexity attack stemming from that interface, or that the challenge program contains a vulnerability at all.

(5) A description of the vulnerabilities in the challenge program.

(6) Inputs or a driver program that demonstrates the vulnerabilities. For side channel vulnerabilities, the driver program should implement the algorithm the AC Team used to compute their operations budget and probability of success. If the adversary needs a notional oracle to complete their exploitation of the side channel vulnerability, include a description of how the adversary would use the oracle, as well.

For practical reasons, the AC Teams may produce challenge programs that use localhost to stand in for network peers that would be remote in an actual deployment. Consequently, the R&D and Control Teams may receive descriptions of benign behavior that mention remote network peers accompanying challenge programs that are coded to use localhost.

3.1.5.2 What Control and R&D Teams Will Receive in Engagements

In engagements, the R&D Teams will receive challenge program binaries and items 1-4 from the list above. The EL will mark each challenge program to indicate whether R&D Teams should check the program for space-based vulnerabilities to algorithmic complexity attack, time-based vulnerabilities to algorithmic complexity attack, space-based vulnerabilities to side channel attack, time-based vulnerabilities to side channel attack, or some combination of these.

The EL Team will provide some additional constraints for the challenge programs with side channel questions. First, they will provide a fixed set of secret symbols. For example, in the context of the health-related system described above the EL would provide a fixed list of health conditions. Second, the EL may also provide some constraints on the secret to be leaked. For example, they might specify that users will search for only one health condition at a time. These

constraints should make it easier for the R&D and Control Teams to reason about the side channel questions.

R&D Teams should expect to receive source and the remaining items after the engagement.

3.1.5.3 What Control and R&D Teams Will Produce in Engagements

For each challenge program, the Control and R&D Teams must indicate whether or not the challenge program contains a vulnerability. If the challenge program contains multiple vulnerabilities of the same type (for example, multiple time-based side channels) it is sufficient for the Control and R&D Teams to report only one of those vulnerabilities.

When indicating the presence of a vulnerability, the R&D Team must provide a justification that explains both the details of the found vulnerability and the details of how the tools supported the discovery of that vulnerability.

For vulnerabilities to algorithmic complexity attack, the justification should include (at a minimum):

1. A reference to the code (one or more sections) that contains the vulnerability,
2. A statement on why it is believed that this code contains a vulnerability (e.g., a statement of the asymptotic order of the vulnerable algorithm, an indication of large coefficient due to a particularly resource consuming operation, etc.)
3. An indication of why it is believed that this code is triggerable by an attacker.

While an example sequence of inputs that causes the code to exceed the resource usage is not necessary, it is a sufficient justification on its own. Note, the underlying vulnerability must still meet the operational definitions provided in this document. As such, witnesses of vulnerabilities must correspond to vulnerability in the provided Java byte code of the application.

For side channel vulnerabilities, the justification should include (at a minimum):

1. A statement on how the observables relate to the secret,
2. A statement on why it is believed that observables reveal the value of the secret (even in the worst case) with a side channel that is sufficiently strong
3. A statement on why it is believed that this side channel is triggerable by an attacker. Note, if the side channel is believed to be passively persistent, a statement to this effect is sufficient.

While an example sequence of inputs and a data analysis algorithm that extracts the secret is not required, it is a sufficient justification on its own. Note, the underlying vulnerability must still meet the operational definitions provided in this document. As such, witnesses of vulnerabilities must correspond to vulnerability in the provided Java byte code of the application.

When indicating the absence of any vulnerabilities, the R&D Team must provide a justification that explains both the details of why they believe the challenge program is free of vulnerabilities (of the type designated in the question) and the details of how the tools supported this conclusion.

For questions regarding algorithmic complexity vulnerabilities, the justification should include (at a minimum):

1. Statements on any potentially vulnerable code structure in the application

2. Statements on investigations of potentially vulnerable code structure including complexities of said code structures
3. Statements on why the application does not contain a complexity vulnerability (strength and reachability).

For questions regarding side channel vulnerabilities, the justification should include (at a minimum):

1. Statements on if and how the observables relate to the secret
2. Statements on any potential side channels within the application

Statements on why the application does not contain a side channel (strength and reachability).

3.2 Common Reference Platform

3.2.1 Introduction

The following sections will detail the configuration of the STAC Reference Platform; consisting of:

- Hardware Platform (NUC)
- Disk partitions and volumes on the NUC SSD
- File System installed on the volumes
- Linux Kernel
- Linux Distribution
- Daemons started by Systemd
- Docker Server
- Docker Containers
- Challenge Programs and STAC Baseline Image

This is followed by section 3.2.11, which provides procedures for performing a reference installation.

Please send any questions related to the reference platform and its installation procedures to `anders.jagd@apogee-research.com`.

3.2.2 Hardware

The hardware Platform will be a NUC, model NUC5i5RYH, with following key specifications:
(http://www.intel.com/content/www/us/en/nuc/nuc-kit-nuc5i5ryh.html)

- CPU
    - Intel(R) Core(TM) i5-5250U CPU @ 1.60GHz
      (http://ark.intel.com/products/84984/Intel-Core-i5-5250U-Processor-3M-Cache-up-to-2_70-GHz)
        - Two 64-bit Cores, each with base operating frequency of 1.6GHz
            - Support Intel Turbo Boost up to 2.7GHz however this will be disabled through BIOS for the reference platform

- Support Intel Speed Step for lower frequency operation
  (will disable frequency reduction through cpupower.service configuration)
- Support Intel Hyper Threading, providing 2 logical processors per Core for a total of 4 logical processors

  - Front Side Bus frequency of 1600 MHz

- Memory Architecture
  - L1 cache (32 KiB I-cache, 32 KiB D-cache per Core)
  - L2 cache (256 KiB per Core)
  - L3 cache (3 MiB)
  - External Memory: 16GiB DDR3L, 1333/1600MHz (64 bit) – max transfer rate of 25.6GB/s
    (Memory Crucial 16GiB Kit 2-8GiB PC3-12800 DDR3 KIT - Model # ct2kit102464BF160B)
- Storage
  - 250 GB SSD (Samsung 850 EVO M.2 Sata 6Gb/s - Model # mz-n5e250bw)
    (see disk partitioning below)
- Network
  - 10/100/1000 Mbps Ethernet
    (Wi-Fi disabled)
    MTU size of loopback interface will be configured as 1500

3.2.3 Disk Partitioning and Logical Volumes

The solid-state drive (/dev/sda) will be partitioned as follows:

- sda1: 200 MiB for EFI, GRUB2
- sda2: 500 MiB for kernel images
- sda3: 68 GiB, managed by LVM2 as 3 logical volumes:
  - centos-root (root filesystem, mounted at '/'): 50 GiB
  - centos-home (user home directories, mounted at '/home/') : 10 GiB
  - centos-swap (swap space) 8000 MiB
- sda4: 40 GiB, managed by LVM2 as 2 logical volumes:
  - vg-docker-data (pool of data for docker images): 35 GiB
  - vg-docker-metadata (metadata for managing above pool): 4.9 GiB
- sda5: Optional 20 GiB "Hot Partition" for installation of Docker Images on "Bare Metal"

3.2.4 File System

XFS Filesystem will be installed on centos-root and centos-home volumes.

3.2.5 Linux Kernel

The Linux kernel will be 3.10.0:

- 64 bit kernel 3.10.0-229.el7.x86_64

- Started as follows by GRUB2:

```
/vmlinuz-3.10.0-229.el7.x86_64 root=/dev/mapper/centos-root ro
rd.lvm.lv=centos/root rd.lvm.lv=centos/swap crashkernel=auto rhgb
quiet LANG=en_US.UTF-8
```

- For this engagement, no restrictions on processor selection will be enforced upon the kernel scheduler. All four logical processors (2 cores, each with 2 logical processors due to hyper threading) may be allocated by the kernel scheduler to each thread, as well as the interrupt handlers.

Detailed kernel configuration may be provided separately.

3.2.6 Linux Distribution

The Linux distribution will be Centos 7:

- CentOS Linux release 7.1.1503 (Core)
- Minimal installation w. Ethernet enabled (DHCP) (as reference, see anaconda-ks.cfg below)
- Systemd target “multi-user.target” (login through ssh or console)
- User ‘stac’ with ‘/home/stac’ home directory and ‘wheel’ group membership (for sudo access)

```
#version=RHEL7
# System authorization information
auth --enableshadow --passalgo=sha512
# Use CDROM installation media
cdrom
# Use graphical install
graphical
# Run the Setup Agent on first boot
firstboot –enable
ignoredisk --only-use=sda
# Keyboard layouts
keyboard --vckeymap=us --xlayouts='us'
# System language
lang en_US.UTF-8
# Network information
network  --bootproto=dhcp --device=enp0s25 --ipv6=auto –activate
network  --hostname=localhost.localdomain
#Root password
rootpw –lock
# System timezone
timezone America/New_York –isUtc
user --groups=wheel --name=stac --
password=$6$O41JeGbbtBigPt6Z$d.lhTowUG.iFIJFC/BnOfiICrBuk57qJK1fbLDNXqtqs2EN0C3
me.Ainliq5yoVd.Nlbl/.Uzn5Xy/sXKy.9T0 --iscrypted --gecos="stac"
# System bootloader configuration
bootloader --append=" crashkernel=auto" --location=mbr --boot-drive=sda
# Partition clearing information
clearpart  --initlabel --list=sda5,sda4,sda3,sda2,sda1
# Disk partitioning information
part /boot --fstype="xfs" --ondisk=sda --size=500
```

```
part pv.695 --fstype="lvmpv" --ondisk=sda --size=69448
part /boot/efi --fstype="efi" --ondisk=sda --size=200 --
fsoptions="umask=0077,shortname=winnt"
volgroup centos --pesize=4096 pv.695
logvol /  --fstype="xfs" --size=51200 --name=root --vgname=centos
logvol swap  --fstype="swap" --size=8000 --name=swap --vgname=centos
logvol /home  --fstype="xfs" --size=10240 --name=home --vgname=centos
%packages
@core
kexec-tools
%end
%addon com_redhat_kdump --enable --reserve-mb='auto'
%end
```

3.2.7 Daemons

The minimal installation includes the following running daemons (after the install script described in section 3.2.11 has installed `docker`, and disabled `firewalld, tuned, crond,Postfix, and NetworkManager`):

```
systemd-journal
lvmetad
systemd-udevd
auditd
irqbalance
rsyslogd
systemd-logind
dbus-daemon
login
sshd
dhclient
docker
```

Note: In future engagements, the `irqbalance` daemon may become configured such as to process interrupts on specific processors.

3.2.8 Docker Server

Docker will be installed on the reference platform:

- Docker version 1.9.1 (docker-engine-1.9.1-1.el7.centos.x86_64.rpm); installed through a script provided by STAC-EL (see section 3.2.11).
- The following Docker configuration options will be set; such as to apply direct-lvm storage (dedicated volumes for storage of docker images; managed as a pool of logical volumes)
    - –storage-driver=devicemapper
    - –storage-opt dm.datadev=/dev/vg-docker/data
    - –storage-opt dm.metadatadev=/dev/vg-docker/metadata
- The following Docker configuration option will be set; such as to apply XFS filesystem for docker images:
    - –storage-opt dm.fs=xfs

- The following Docker configuration option will be set; as workaround for a bug in docker related to Systemd management of Docker cgroup settings (the default w/o this option):
    - –exec-opt native.cgroupdriver=cgroupfs
- The Linux user 'stac' will be added to the Linux group 'docker', such as to allow execution of docker commands without sudo.

3.2.9 Docker Containers

Challenge Programs will be managed as Docker Containers, hosted by a STAC Docker Registry installed on the reference platform. Challenge Programs will be pulled and run from this STAC Registry through a script (`manage_stac_docker_registry_<version>.sh`).

- Challenge Programs will be run within Docker Containers; started with the following options:
    - –net=host (in order to disable NAT'ing between Docker container and localhost)
- No (cgroup) restrictions will be enforced on Docker Containers for access to the resources of the reference platform.

3.2.10 Challenge Programs and STAC Baseline Image

Challenge Programs will be Java bytecode, run within a Docker Container consisting of an XFS filesystem with the following layers:

- STAC Base Image:
    - Centos 7.1.1503 (tag centos:7.1.1503 from `hub.docker.com`)
        - No services/daemons running within container unless installed by Challenge Program
    - Java Runtime Environment java-1.8.0-openjdk-1:1.8.0.65-2.b17.el7_1.x86_64.rpm (available at http://vault.centos.org/7.1.1503/updates/x86_64)
- Challenge Program and its dependencies
    - Challenge Programs will be run with JIT enabled

3.2.11 Installation Walk-through

The following describes procedures for performing a STAC Reference Installation on a fresh NUC:

1) Connect NUC to a monitor, keyboard, mouse, and Ethernet (with DHCP server reachable on the network)

2) Disable Intel Turbo Boost through BIOS Settings (boot + f2)

3) Install CentOS 7.1.1503 from Minimal Installation CD/DVD, as downloaded from e.g.:

```
http://vault.centos.org/7.1.1503/isos/x86_64/CentOS-7-x86_64-Minimal-1503-01.iso
```

NOTE: After powering up the NUC, press F10, then select the **UEFI** Bootloader option from the CD/DVD.

Apply following selections:

- Select "Minimal Install" with no Add-Ons
- Select "I will configure partitioning", do not select “Encrypt my data”
- Configure the following partitions:
    - /boot/efi
        - Mount Point /boot/efi
        - Device Type "Standard Partition"
        - File System "EFI System Partition"
        - Desired Capacity 200MiB
    - /boot
        - Mount Point /boot
        - Device Type “Standard Partition”
        - File System XFS
        - Desired Capacity 500MiB
    - /
        - Mount Point /
        - Device Type LVM
        - File System XFS
        - Volume Group centos
        - Name root
        - Desired Capacity 50 GiB
    - /home
        - Mount Point /home
        - Device Type LVM
        - File System XFS
        - Volume Group centos
        - Name home
        - Desired Capacity 25 GiB
    - Swap
        - Mount Point: None
        - Device Type LVM
        - File System swap
        - Volume Group centos
        - Name swap
        - Desired Capacity 8000 MiB
- Select following network settings:
    - Ethernet ON
    - Wireless OFF
- Create stac user:
    - Full name stac
    - User name stac
    - Make user administrator

- Require a password to use this account
- Password stacme (click Done twice since password is weak)

4) Following CentOS installation, reboot and login as stac. Then determine the IP address assigned by DHCP by typing "`ip addr list`"

5) Download the following "STAC Reference Platform Configuration" script, found in the STAC GIT Repository "Public_EL_Information", to the NUC (e.g. using scp):

```
stacReferencePlatformInstall.v2.3.sh
```

6) From the NUC (console or through ssh connection), run the script:

```
sudo ./stacReferencePlatformInstall.v2.3.sh all
```

The script will remove some daemons, install the `cpupower Systemd` service, and finally download and install the `docker` service.

7) Reboot the NUC

3.3 Common Reference Network Environment

For Engagement 5 onward, we will be moving from the current localhost implementation of all server and client interactions. This section provides configurations for the reference network environment as well as an example implementation of the reference network configuration to be used by Apogee for testing future challenge programs. The reference network environment components are as follows:

- 2 NUCs running the reference platform established in sections 1 through 11 of this document. (1 server device, 1 client device)
- 1 Netgear GS108Ev3 managed switch
- 1 NUC running the reference platform for observing packets on the mirrored port and acting as attacker when required.
- Optional: 1 device for managing and monitoring the switch (we use a laptop running windows 10)

Given that the specs of the refence network environment have been enumerated we will proceed to setup the switch.

3.3.1 Switch Configuration Walk-through

The Netgear GS108Ev3 switch can be configured by navigating to the device IP address. Ensure that the switch is not connected to a DHCP server during the entire configuration process. To access the web-management interface:

1) Connect a device with a browser to any of the ports on the switch, and navigate to the default static IP address (192.168.0.239). Before you can navigate to the switch address, the network interface card (NIC) connected to the switch must have the same 192.168.0.x IP address name and the same 255.255.255.0 subnet mask. Instructions for implementing these settings are as follows:

a. Windows 10 OS:
    i. Control Panel → *Control Panel\Network and Internet\Network and Sharing Center*
    ii. Right click on network device connected to switch and click on *properties* tab
    iii. Click on *Internet Protocol Version 4 (TCP/IPv4)* and click on *properties* tab
    iv. Set IP address to *192.168.0.90* and subnet mask to *255.255.255.0*. Leave default gateway field empty.
b. CentOS 7:
    i. Navigate to */etc/sysconfig/network-scripts*
    ii. Identify the network interface config file (e.g *ifcfg-enp0s25*) and save a copy before proceeding
    iii. Ensure that the following settings are changed:
        1. *TYPE="Ethernet"*
        2. *BOOTPROTO="static"*
        3. *IPADDR="192.168.0.90"*
        4. *NETMASK="255.255.255.0"*
    iv. Run *sudo ifdown <device name e.g enp0s25>*
    v. Run *sudo ifup <device name e.g enp0s25>*

2) Use the default username *admin* (may not be required) and default password *password* to log in.
3) Upgrade to firmware version 2.00.09
    a. Download the firmware upgrade from: http://www.downloads.netgear.com/files/GDC/GS108EV3/GS108Ev3_V2.00.09.zip
    b. Unzip the file
    c. In the web interface navigate to the *System* tab then the *Maintenance* tab
    d. Click on the *Firmware Upgrade* link
    e. Click on the *Enter Loader Mode* link
    f. Click on the *Firmware Upgrade* link and browse to the *GS108Ev3_V2.00.09.bin* file extracted from the *GS108Ev3_V2.00.09.zip* file
    g. Wait for the firmware upgrade to complete and the switch to reboot
4) Log in with the default password *password* to the web-management interface
5) Proceed to the *System* tab then the *Maintenance* tab
6) Download the *GS108Ev3.cfg* file from the *Public_EL_Information* repo in the *Reference_Network_Environment* folder.
7) Click on the *Restore Configuration* link and browse to the *GS108Ev3.cfg* file
8) Wait for the settings to be loaded and the switch to reboot. The updated settings are as follows:
    a. Change of the Switch Name to *STAC Switch*
    b. DHCP Mode Disabled
    c. Change of the switch IP address to 192.168.100.50
    d. Change of the switch subnet mask to 255.255.255.0
    e. Change of the switch gateway address to 192.168.100.1
    f. Change of the port mirroring configuration to mirror port 2 and port 4 traffic on port 3

Note: the port mirroring configuration is required for the reference network environment; however, the specific ports used and the specific assigned IP addresses, subnet mask, and gateway address are not required. Teams may choose to use different values but these are the values and specific configurations that will be used to test engagement challenges.

9) Use the instructions in "1)" to change the IP address of the device connected to the switch to a *192.168.100.x* name e.g *192.168.100.90* and ensure that the device is connected to a port other than 1, 2, 3 or 4.
10) Log in with the default password *password* to the web-management interface and confirm that the settings in "8)" are applied.

3.3.2 Network Configuration

As stated in the previous section, the reference network environment calls for 2 NUCs (*serverNuc* and *clientNuc*) running the reference hardware, a third NUC running the reference hardware (*masterNuc*) to be used for monitoring traffic on the mirrored port (set as ports 2 and 4 traffic mirrored to port 3 for EL configuration), and a Netgear GS108Ev3 switch running firmware version 2.00.09 and the configurations specified in the *GS108Ev3.cfg*. The device configuration used by the EL is as follows:

Table 2: Network Configuration

Device	Static IPv4 Address	Switch Port Number
Reference NUC (*serverNuc*)	192.168.100.20	1
Reference NUC (*clientNuc*)	192.168.100.30	2
Mirrored port monitor NIC (*masterNuc*)	NA	3
Attacker NIC (*masterNuc*)	192.168.100.10	4
Optional Switch setup/maintenance PC	192.168.100.90	8
Netgear GS108Ev3 Switch	192.168.100.50	NA

To minimize the chance of confusing the different devices and their IP addresses, the EL modify the */etc/hosts* file for the three devices to map the static IP addresses to a descriptive name of the device's function. The following lines are added to the *etc/hosts* files of the *serverNuc, clientNuc,* and *masterNuc* devices:

192.168.100.10 masterNuc
192.168.100.20 serverNuc
192.168.100.30 clientNuc

As stated above, *masterNuc* is a NUC running the reference platform. Since the reference platform only has one NIC, we have attached a USB-to-Ethernet device (*StarTech USB 3.0 to Gigabit Ethernet NIC* model number: *USB31000SW*). The USB-to-Ethernet device (NIC #1) is the NIC for collecting data on STAC observables (e.g. timing, packet sizes etc.). The internal

NIC (NIC #2) which came with the NUC is used as the attacker when an active attacker is required.

To allow for the mirrored port NIC (NIC #1) to capture all traffic, the device must be set to *promiscuous mode*. Instructions to setup *promiscuous mode* in Windows 10 are as follows:

- Assuming the use of *Wireshark* to collect packet data, under *Capture → Options*, ensures that *Promiscuous* is *enabled* on the interface to be used for capture

Instructions to setup *promiscuous mode* in CentOS 7 are as follows:

- In command line enter: *sudo ip link set <device e.g. enp0s20u4> promisc on*

Note: in the EL's configuration, the mirrored port NIC has the device name: *enp0s20u4*. The proofs for E5+ engagements (provided to Blue Teams after the engagements) will assume a device name of *enp0s20u4*. When the device is connected, in CentOS, the device is assigned a device name that may or may not be the same as *enp0s20u4*. To change the name of the mirrored port NIC:

1) Run *ip addr list* on the *masterNuc* device where the NIC is connected and note the assigned device name. If another NIC has the *enp0s20u4* the instructions below can be used to change that device name.
2) Navigate to */etc/sysconfig/network-scripts* identify the network interface config file (e.g *ifcfg-enp0s25*) for the mirrored port NIC.
3) Open the file and change the *NAME=* and *DEVICE=* options to *NAME="enp0s20u4"* and *DEVICE="enp0s20u4"*
4) Change the network interface config filename to *ifcfg-enp0s20u4*
5) Run *sudo ifdown enp0s20u4*
6) Run *sudo ifup enp0s20u4*

3.3.3 Setup and Observing Vulnerabilities

Sections 12.1 and 12.2 establish the reference network environment requirements and provide an example implementation. This section provides further details on the setup of the network environment and on how vulnerabilities, as specified in the E5+ operational definition document will be measured.

The reference network environment has 2 reference NUCs as *serverNuc* and *clientNuc*. *serverNuc* runs the application we are interested in monitoring for resource consumption. *masterNuc* has two NICs, one listening on the mirrored port of the switch and the other which will, in cases where an active attacker is needed, be used to send the attack to the *serverNuc*.

NIC #1 (connected to the mirrored port) collects data from both the active attacker's interaction and from the benign client's interaction. For algorithmic complexity attacks, the attacker does not have access to the data collected on NIC #1. The data collected on NIC #1 will be used to confirm the success of a remote-DoS AC Time attack (see OpDef). The success of an AC Space attack will be confirmed via an observable (e.g. disk space, RSS memory usage etc.) on *serverNuc*. The success of a self-DoS AC Time attack will continue to be confirmed via the time from the command submission to the applications response measured on *serverNuc*.

For side channel attacks, the attacker has access to the data collected on NIC #1. Note: all observables available for side channel challenge questions must be observed through NIC #1.

The attacker does not have access to clientNuc and the only allowed access to serverNuc is to measure space resource consumption for AC Space challenge questions.

Note: The mirrored switch configuration results in the packets observed via the mirrored port having 2 extra trailer bytes than the packets observed on the sender or receiver. This is because port mirroring appends information on the source network switch port for the mirrored traffic.

The figures below demonstrate the setup of the reference network environment.

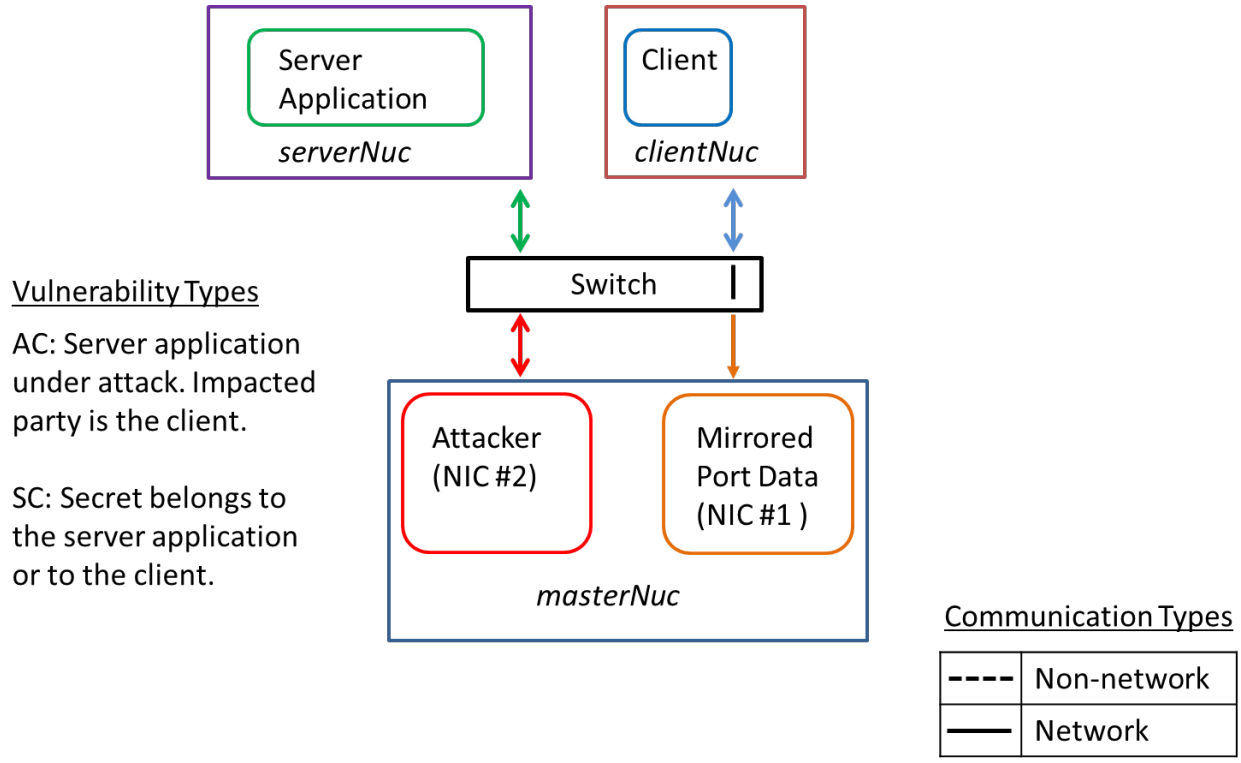


Figure 6: Client-Server Setup of Reference Network Environment

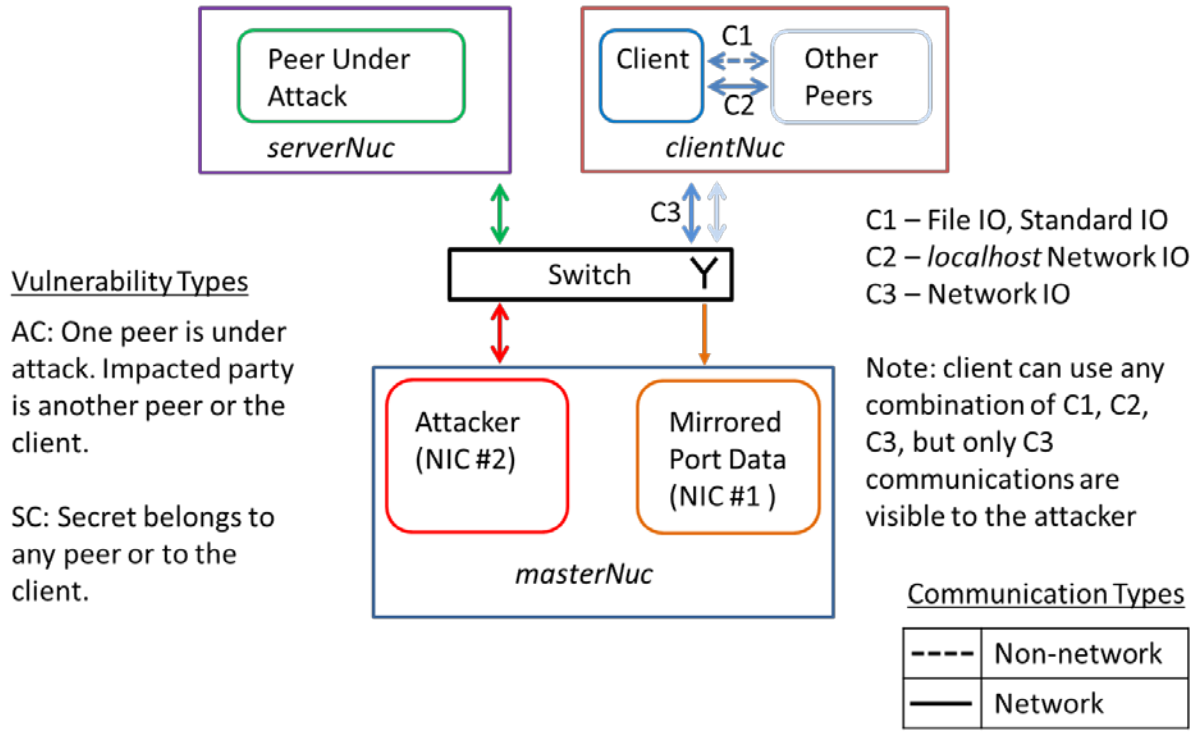


Figure 7: Peer-to-Peer Setup of Reference Network Environment

3.4 Common Reference Platform Analysis

3.4.1 Docker Experiments and Analysis

Docker provides a convenient environment for organizing challenge problems. Local Docker Repositories allow sets of challenge problem images to become compactly organized and distributed for Engagements. At runtime, Docker containers allow challenge programs to be presented in consistent environments, tailored to each individual challenge. Conflicts between challenge programs due to any residual effects of (partial) de-installation of individual challenge programs and their dependencies are eliminated.

Docker containers execute processes and perform system calls directly towards the kernel. As such, Docker is not a hypervisor; however, the potential still exists for differences in timing between running a process on the bare metal reference platform and running a process inside a Docker container. The potential differences can be reduced to:

1. Pure CPU based with no I/O
2. Standard I/O
3. File I/O
4. Network I/O

Three test programs were used to analyze the impact of Docker:

- Test Program 1 contains a time-based algorithmic complexity vulnerability. It manages key and value pairs using a hashing function followed by linked link traversal, and as

such exhibits difference between fast and slow performance based upon frequency of collisions.

- Test Program 2 has a size-based side channel vulnerability. It concatenates user input with the contents of a (secret) file, then compresses and encrypts the result.
- Test Program 3 contains a different time-based algorithmic complexity vulnerability. It runs an algorithm to calculate the ratio of similarity between two input strings based upon recursive search for “longest common substrings”.

3.4.1.1 Pure CPU with No I/O

In the absence of I/O operations, and without deliberately imposing restrictions on memory, CPU, block I/O, or any other system resources (through the Docker use of Linux cgroups), timing differences between running a process inside a Docker container or on the bare metal common reference platform, should be negligible compared to any reasonable separation between “fast” and “slow” execution of a challenge problem. To verify this, Test Program 1, which is dominated by pure CPU pipeline throughput, was applied. Program 1 was run with the standard output from the program redirected to /dev/null; this minimize effects of STDIO redirection between the executing host shell and a Docker shell executed within the container.

The result of the test (Figure 8) shows that the program runtimes on the common reference platform and Docker are consistent; as compared to the difference between fast and slow input.

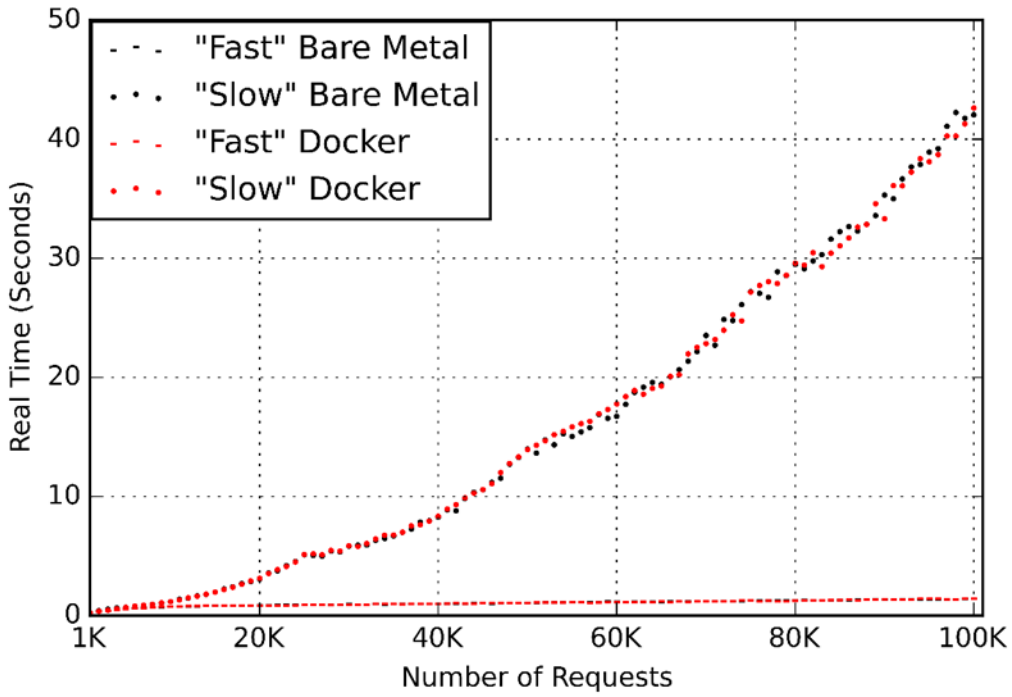


Figure 8: Test Program 1 with STD I/O redirected to /dev/null. Consistent runtimes on Bare Metal and Docker

3.4.1.2 Standard I/O

To observe the expected impact of redirection of standard I/O, Test program 1 was also run without redirection of the program output to /dev/null. The results, shown in Figure 9, indicate that the decrease in speed on Docker is linear with the number of requests (each request having the same amount of characters presented on STDIN and STDOUT).

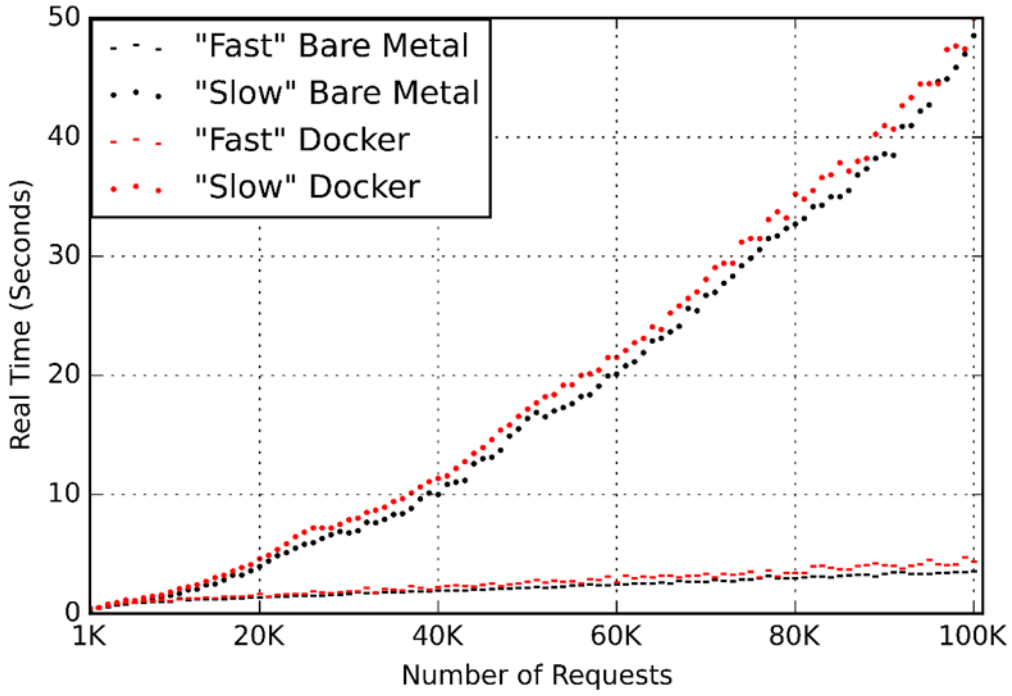


Figure 9: Test Program 1 with STD I/O. Slowdown is linear with number of requests

For this case, the difference between the Docker and Bare Metal times (Figure 10) is approximately 7.4 ± 0.576 µs*[number of requests] + 17.23 ± 33.5 ms in the 'fast' case; with remaining clear separation between 'fast' and 'slow' cases. For weak side channels, such timing difference could potentially present an issue; this however mitigated by the timing difference being consistent for each request, allowing the side channel to still be observable – albeit with reduced signal to noise ratio.

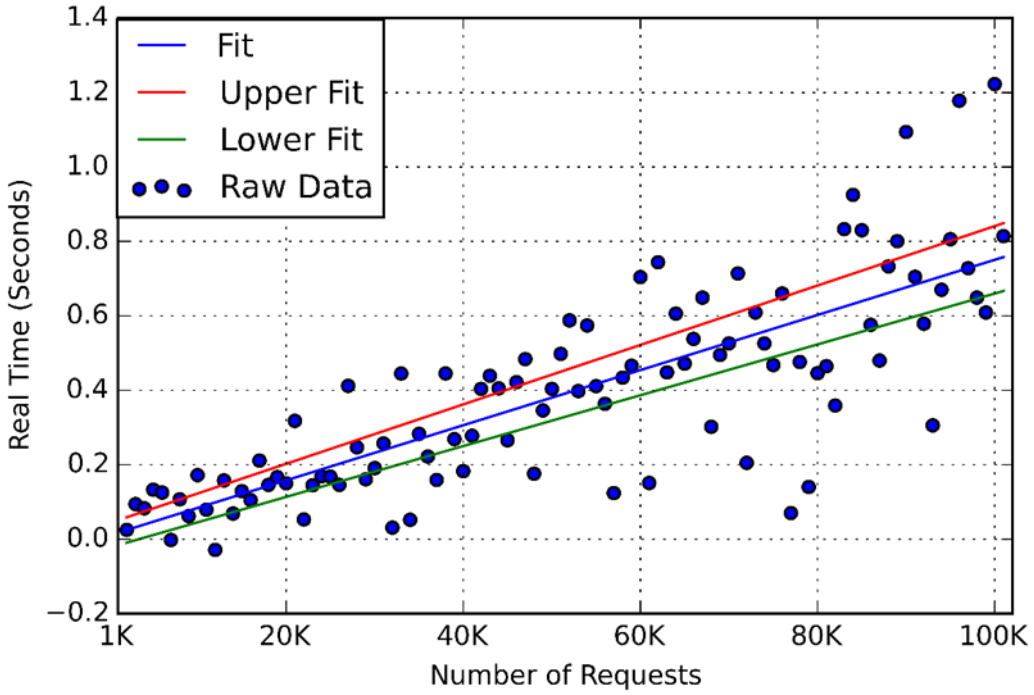


Figure 10 Linear fit of difference between Docker and Bare Metal

3.4.1.3 File I/O

3.4.1.3.1 Example 1: Test Program 2

Test program 2 is not intended as an example of a time-based vulnerability. However, since it exhibits heavy use of file I/O, it is suitable for testing timing differences introduced by variations in the File I/O mechanics between a running Docker container and the common reference platform. Docker was configured with the same filesystem as the reference platform, but subtle differences related to how Docker manages the volumes may introduce non-zero timing differences.

Test Program 2 was invoked 1 to 100 independent times using the same secret file and appended input. The File I/O test results (Figure 11) indicate that Docker introduced an additional lag of 13.275 ± 0.103 ms *[Number of requests] +1.63 ± 5.99 ms; as compared to the common reference platform. Because Docker uses a Thin device mapper scheme and the standard install of Cent-OS uses a Linear device mapper scheme, we also ran a Thin device mapper on bare metal to ensure that it doesn't create a difference.

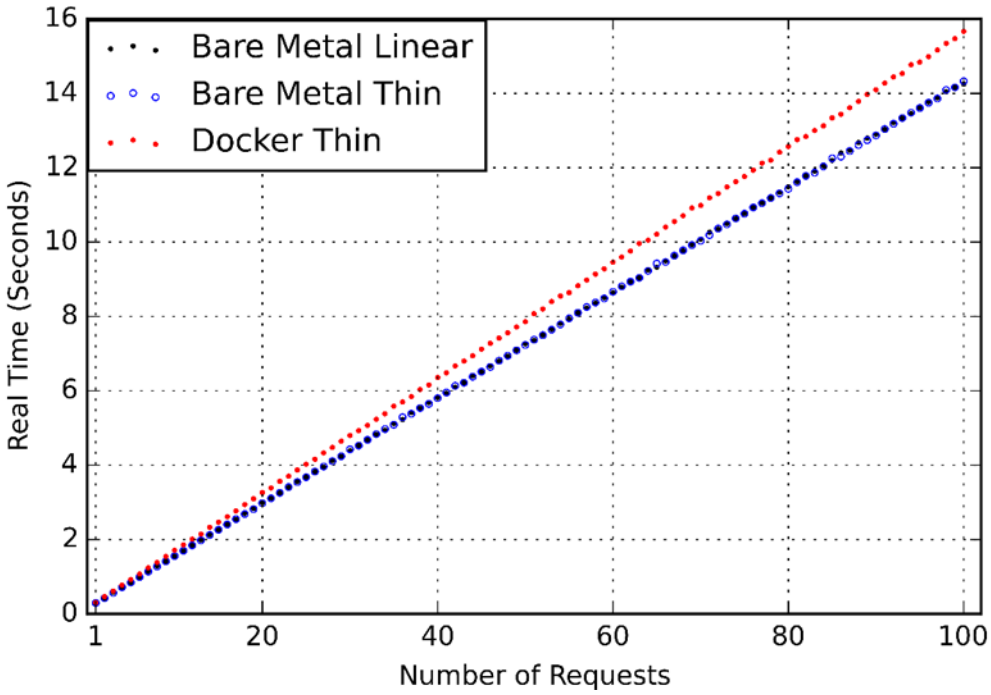


Figure 11: Test program 2 File I/O results. Slowdown is linear with number of requests.

3.4.1.3.2 Example 2: Test Program 3

Test Program 3 exemplifies a time-based algorithmic complexity vulnerability; and also, incorporated file I/O in order to generate (write) test input and then read the same in order to calculate the similarity index. Test program 3 was run on both the common reference platform and on Docker.

The results from Test Program 3 (Figure 12) show a timing difference between Docker and Bare Metal of -11.25 ± 484 µs*[number of characters (bytes)] – 0.16 ± 1.4 ms in the 'fast' case. In this example Docker was faster than Bare Metal. This is something to be considered for engagements with weak time-based side channels.

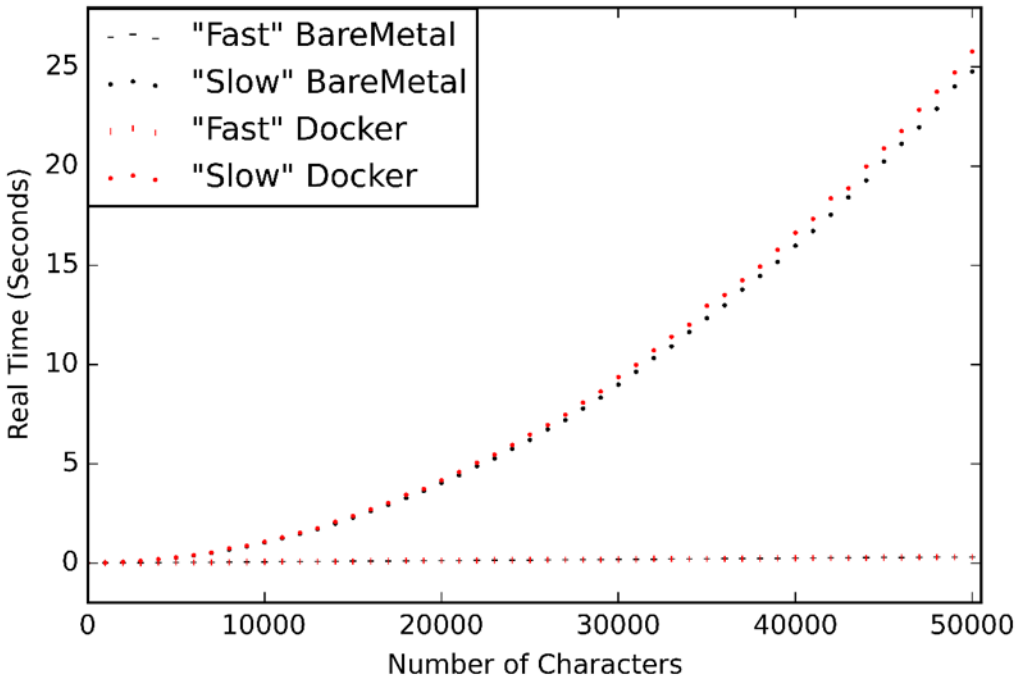


Figure 12: Test Program 3 file I/O results; linear slowdown with number of requests

3.4.1.4 Network I/O

3.4.1.4.1 Example 1: TCP latency per packet

To test network I/O differences between Docker and the common reference platform, two sets of experiments were conducted.

The first set of experiments established a TCP/IP connection from an (iperf) client to an (iperf) server, and applied this connection for repeated transmission of 8kB buffers over TCP. The client and the host were run on the same (NUC) host (i.e. localhost networking). As such, we removed any pipelining effects of multiple packets potentially being concurrently en-route over an actual wire/network. This allows us to estimate the end to end transfer latency of individual packets directly from the packet throughput measured by iperf.

Under such circumstances, the per-packet latency is expected to be dominated by the effects of sandboxing network traffic within a Docker container through (optional) Netfilter port NAT'ing. To verify this, the server was always run as a regular process (no Docker), while the client was run both from outside a container, as well as within a Docker container; with and without NAT.

The results are seen on Figure 13 (iperf results seen to the right are expressed as average throughput with standard deviation over 10 runs).

Figure 13: Network I/O throughput results

The MTU size for localhost networking was set as 64kB, so each 8kB buffer would have been transmitted in a single IP packet. The per packet latency without NAT'ing can therefore be estimated as 8kB/packet/(30Gbps) = 2us, while latency with NAT'ing increases to appr. 8kB/packet/(11Gbps)=6us – for an approximate overhead per packet introduced by Docker/NAT'ing of 4us. Without NAT'ing ("Docker Direct"), as expected, the Docker is seen not to introduce any measurable latency impact.

3.4.1.4.2 Example 2: Test Program 3 with Networking

A second networking test was performed using Test Program 3. The program was modified to send and receive strings for comparison using TCP stream sockets with a 1024 byte receive buffer size. Two cases were tested: switched Ethernet (100 Mbps); localhost networking. Both cases were conducted using: a server socket on the common reference platform; and a server socket running in a Docker container with no NAT.

The test results (Figure 14 and Figure 15) again show that the runtimes for test program 3 are consistent on both platforms over Ethernet and over localhost networking.

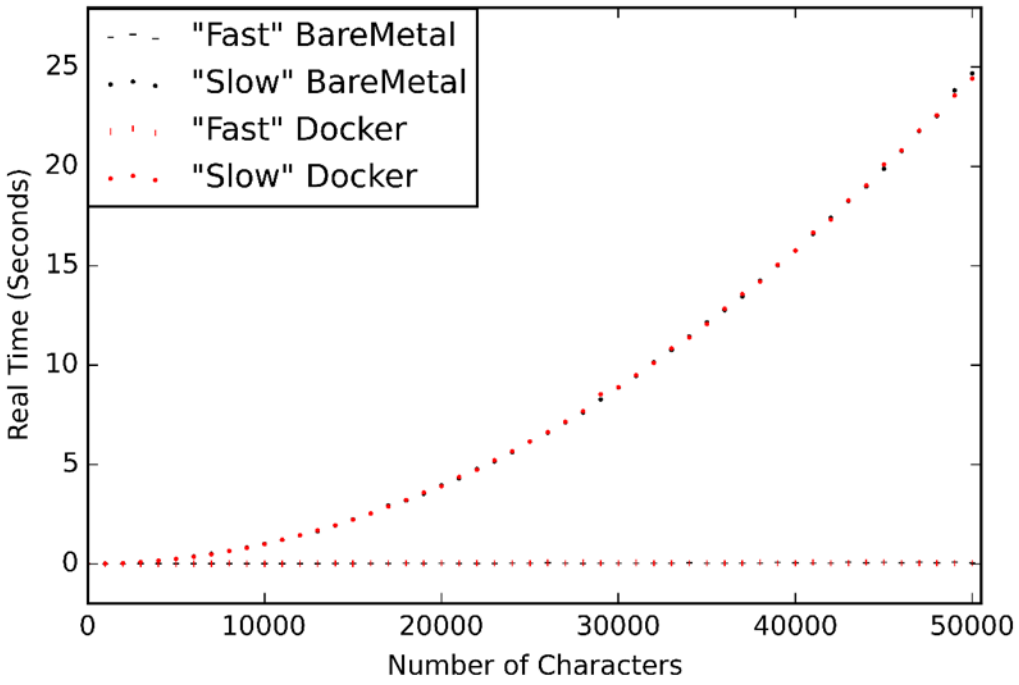


Figure 14: Network I/O TCP packets over Ethernet; consistent runtimes in both cases.

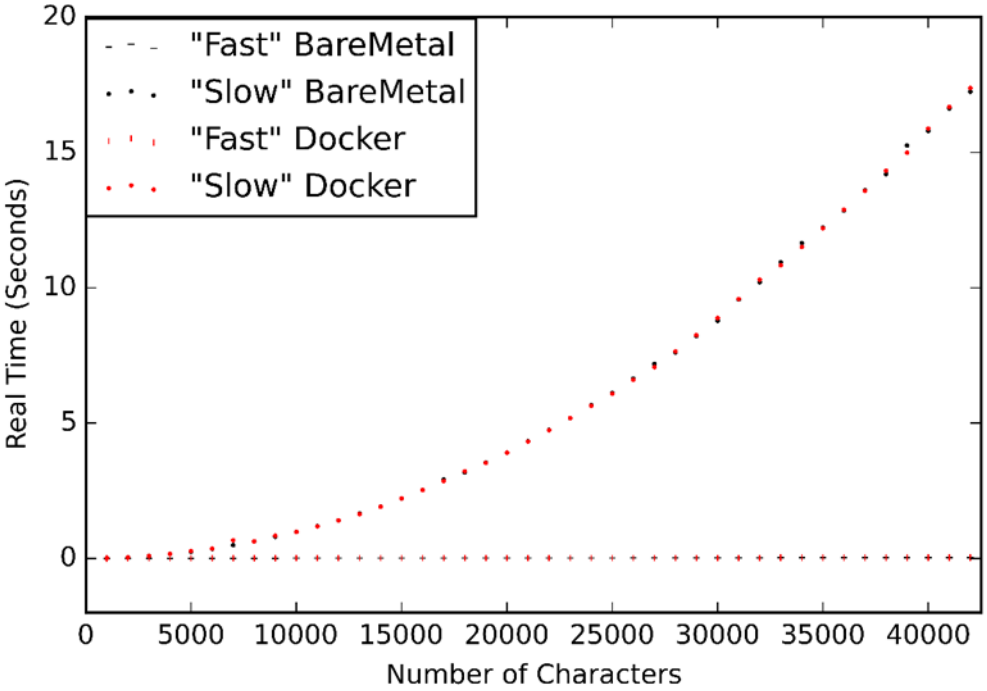


Figure 15: Network I/O TCP packets over Localhost; consistent runtimes in both cases.

3.4.1.5 Conclusions on Docker's Impact

The test results show that Docker, configured without restrictions on platform resources (CPU, memory, block I/O, etc.), has a measurable effect on program runtimes. The file IO case showed the most observable impact of Docker. There is a linear slowdown expected with respect to the number of IO events. This difference should not impact engagements involving time-based algorithmic complexity vulnerabilities. Engagements with time-based side channels vulnerabilities will however have to be closely observed when running inside Docker containers to ensure that the side channels are still observable to the same (or similar enough) extent as when running on a bare metal reference platform.

We are evaluating alternative techniques to further minimize this effect and will continue to post information as our analysis proceeds.

3.4.2 Reference Network Environment Testing

The final reference network environment configuration for STAC is detailed in section 12 of the *STAC-EL Reference Platform v3.0* document. Using this configuration, we performed tests to analyze the impact of the refence network environment on timing. There were three tested network configurations:

1. Localhost – the client, server, and timing sensor all run on same device (E1 – E3 network environment)
2. Network: Client – the client and server run on different devices with the timing sensor on the client device.
3. Network: Mirror – the client and server run on different devices with the timing sensor on an external device listening via a mirror port of the client device port on the switch.

There were 3 tested server configurations:

1. Simple server – server receives a request, waits 2.5 ms via a thread sleep and replies to the request.
2. Password authentication server – server receives a request, verified the password via a segmented password authentication algorithm and replies with the authentication result. The authentication algorithm incurs a delay of 2.5 ± 0.25 ms for each correct character.
3. Law Enforcement Database (LED) server – server is the LED challenge program from E1/E2. The server receives a request for a search ranges and proceeds to query the backing database for a list of public and private keys in the search range. The server iterates through the list of keys and returns the key is a single UDP packet if it is public and throws and handles an error if the key is private. The gap between subsequent UDP packets leaks information about whether there are any private keys in the gap between two public keys.

1000 samples were collected for each network environment and server combination. The first set of results for the simple server is shown below.

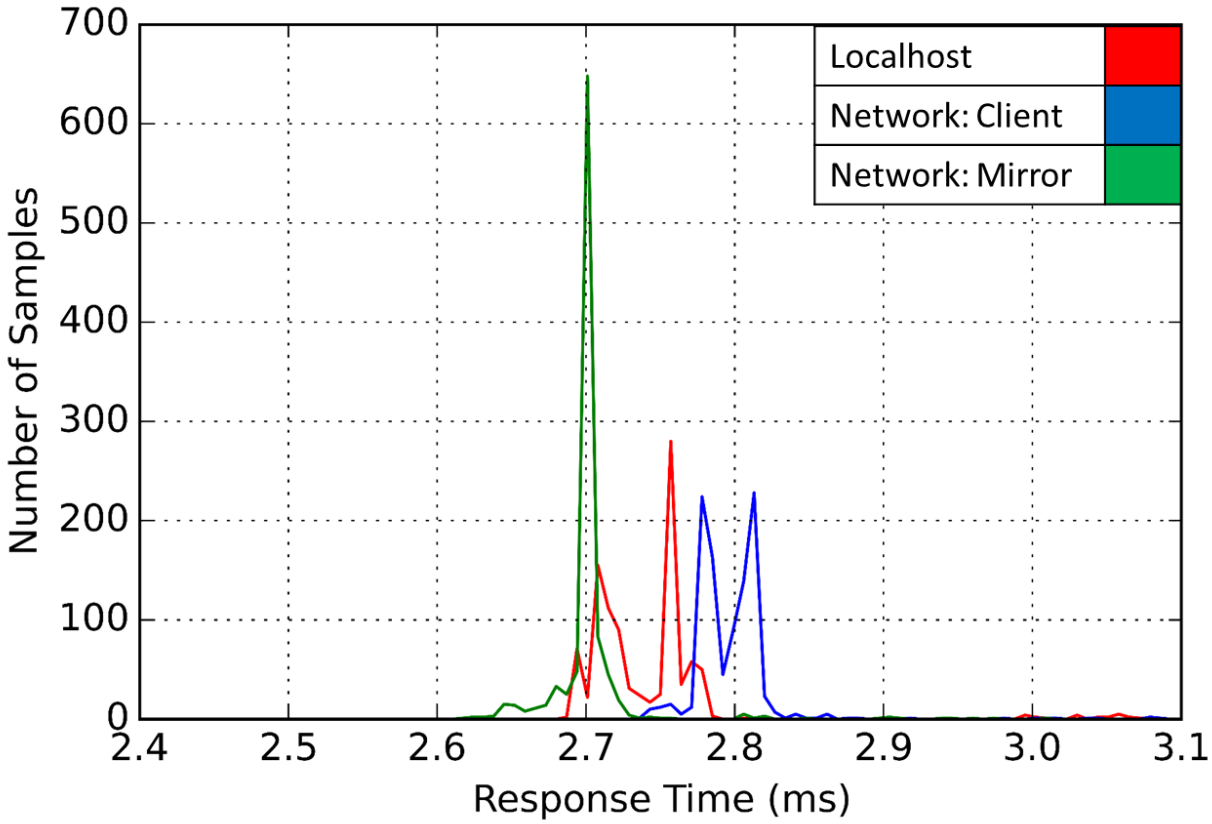


Figure 16: Simple server results

In Figure 16, in the localhost network configuration, there are 3 arguably 4 peaks in the response time distribution. In the network client timing configuration, there are two peaks in the response time distribution. The network mirrored timing configuration resulted in a response time distribution with a single dominant peak. In the localhost and network client configurations, the multiple peaks in the response time distribution are likely due to resource competition between the different applications running of the same system.

Figure 17 below shows the response times for the password authentication server.

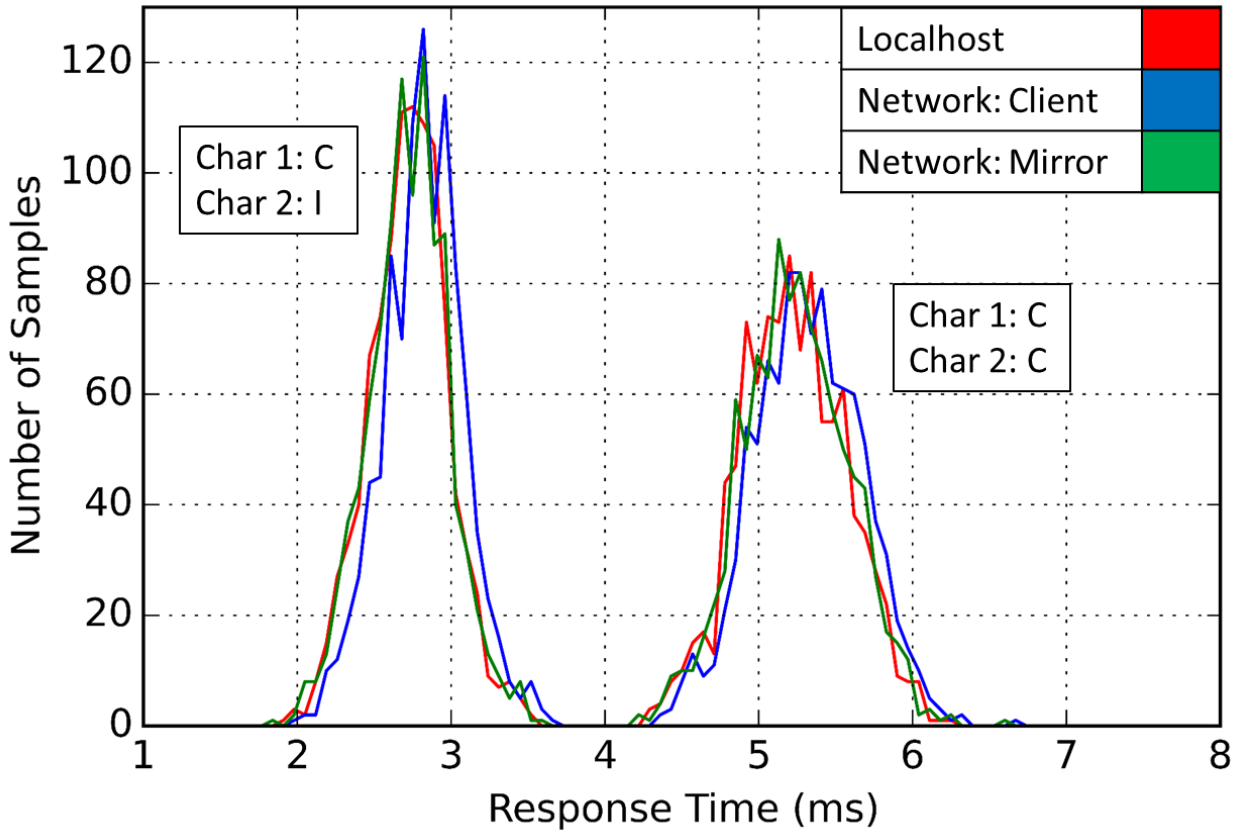


Figure 17: Password authentication server results

In the password authentication example, the side channel is sufficiently strong in all three network configurations. The response time distribution for the network client configuration shows a slight shift to the right in comparison to the other network configurations. This is consistent with the understanding that the times measured on the mirrored port are at the top of the network stack i.e. the packets are on the switch in comparison to the network client configurations where timing is collected further down the network stack. The localhost configuration does not have the same network impact as the network client configuration hence it is expected that the response time would be more similar to the network mirrored timing.

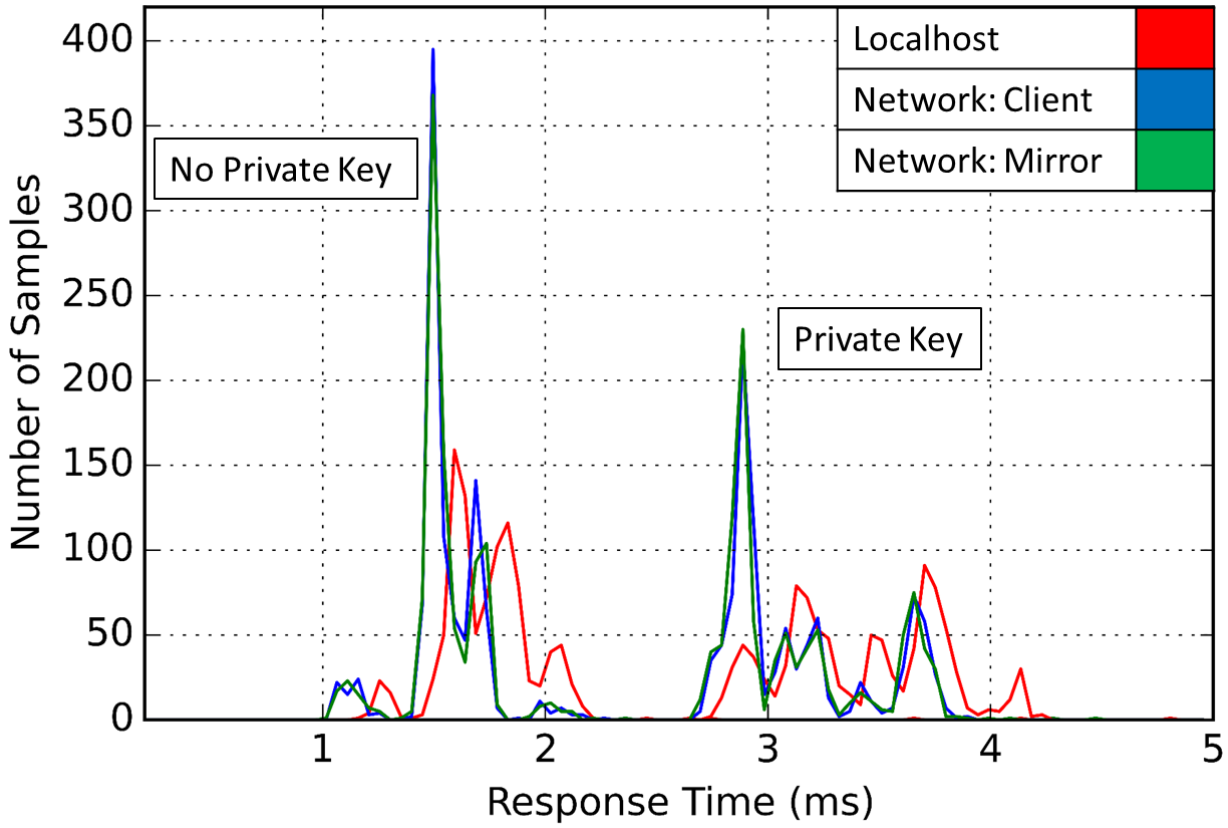


Figure 18: Law Enforcement Database server results

Figure 18 shows the results for the Law Enforcement Database server. This challenge application performs file I/O to strengthen the side channel. This experiment shows multiple peaks in the response time distribution for all 3 network configurations. The localhost network configuration however has more peaks and a shifted response time distribution in contrast to the network client and mirror configurations which appear almost identical. The increase in response times in the localhost configurations is most likely due to resource competition between the server, client, timing instrument (tcpdump) and timing application.

Prior to conducting the experiments, it was our expectation that the introduction of a reference network configuration and a move from the current localhost implementation would add noise to measurements and potentially introduce some real-world issues to exploiting side channels e.g. delay, latency, packet loss, and potential impacts of Nagle's algorithm. What we observed from the experiments, and perhaps should have expected, was that by minimizing competition for resources with the server on one machine, the client on another, and the timing instrument on a third machine, the timing distributions observed on the network mirrored pot are cleaner.

4 Results and Discussion

This section of the report provides a more detailed look at the engagement plan for each of the 7 STAC Engagements, along with the results of the engagement and the assessment of the participating performers. Note the detailed responses provided by each performer have been omitted from this section and are instead covered in Appendices A and B.

4.1 Engagement 1

4.1.1 Engagement Plan

4.1.1.1 Purpose and Program Scope

4.1.1.1.1 Program Scope

This Engagement Plan is for Engagement 1.
The document is structured into the following sections.

Section 1: Purpose and Program Scope
Section 2: Operational Definitions and Research Hypotheses
Section 3: Engagement 1 Plans and Logistics
Section 4: Post-Engagement Analysis Plans

4.1.1.2 Operational Definitions and Program Hypotheses

4.1.1.2.1 Operational Definitions of STAC Vulnerabilities

As a means to guide the Engagements, the STAC AC, EL, and government team has come up with operational definitions of vulnerability to algorithmic complexity attack and to side channels. The AC performers will use these definitions to determine when the vulnerabilities they've inserted into their challenge programs are severe enough for use in engagements. The Control and R&D Teams will use these operational definitions to determine when they've found a vulnerability worth reporting – i.e., if it meets the definition of a STAC vulnerability. [The operational definition document specifies the definitions for STAC vulnerabilities along with guidance for Blue Team responses to engagement challenges.]

4.1.1.2.1.1 Unintended Vulnerabilities in Challenge Programs

Many Algorithmic Complexity and Side Channel vulnerabilities may exist in each challenge program beyond those intentionally inserted. For example, a normal authentication function is technically a side channel that leaks a secret. However, most of these vulnerabilities are "weak" – e.g., for Algorithmic Complexity, the additional complexity is a small percentage of the normal complexity, and for Side Channels, the number of operations necessary to resolve the secret is impractically large.

The defined operational definitions defined above are designed to minimize these unintended vulnerabilities in the challenge programs with the use of strong vulnerability thresholds (e.g., adversary operation budget is small for Side Channels). These thresholds will serve to discount most of these "unintended" vulnerabilities as not meeting the STAC definition. However, some unintended vulnerabilities will occur that do meet the definition. A reported actual vulnerability that satisfies the operational definition criterion for a given challenge program is a correct

response, regardless if it is an intended or unintended vulnerability on behalf of the AC teams. During the analysis period after each Engagement, the EL team will work with the R&D and Control performers to verify that reported unintended vulnerabilities do actually meet the operational definition so that full credit is given.

4.1.1.2.2 Research Hypothesis and Segmentation of Challenge Problems

Throughout the STAC program, the EL team will have a set of research hypotheses pertaining to performance of the tools with respect to different types of STAC vulnerabilities and challenge problems. Example hypotheses include:

1. Complexity vulnerabilities can be due to: (1) a change in the complexity of the algorithm, (2) a change in the coefficients of a fixed complexity, or (3) a flawed data guard that allows processing of input data that shouldn't be allowed. How do the different R&D techniques perform with respect to each of these types of complexity vulnerabilities?
2. Side channels can be weakened by increasing the number of secret symbols that are mapped to identical observed symbols or by increasing the width of the parent distributions of the observed symbols (causing overlaps of the of the samples from different observed symbol distributions). How do the different R&D techniques perform with respect to each of these challenges?
3. Some non-vulnerable programs are locally balanced (e.g., all paths through a sub-section of the binary consume equivalent time and space resources) while others are non-locally balanced (e.g., resource consumption is asymmetric on a set of paths in multiple parts of the program resulting in entire execution traces that are balanced). How do the different R&D techniques perform with respect to the locality of balancing of resource consumption?

In support of these hypotheses, each challenge problem will be assessed and categorized with respect to how results on the challenge problem will support resolution of each of the hypotheses. This segmentation of the challenge problems will not be exposed during the engagements, but will be tracked for post-engagement analysis.

During the engagements, each challenge problem will come questions such as:

1) Does the challenge problem contain a time-based side channel?
2) Does the challenge problem contain a space-based side channel?
3) Does the challenge problem contain a time-based complexity vulnerability?
4) Does the challenge problem contain a space-based complexity vulnerability?

Note, that the questions will be more specific version of these classes of questions including additional context information and could include questions regarding combined time and space vulnerabilities

4.1.1.3 Engagement 1 Plans and Logistics

4.1.1.3.1.1 Engagement 1 Schedule

The schedule below provides an overview of the milestones leading up to Engagement 1, including preparation activities, the live engagement itself, and the post engagement evaluation.

- Release Engagement Plans (EL team): Initial release on 10/1/2015 with approximately monthly updates. Post on GitLab Public_EL_Information Project.
- TA 1 Software Tools (R&D teams): Initial release on 10/15/2015. Post on GitLab individual R&D Project.
- Candidate Challenge Programs (AC teams): Initial release on 11/1/2015 with final release on 12/15/2015. As this is an initial release, there isn't an expectation that all Challenge Programs will be complete, but at a minimum, all expected Challenge Programs should be described. Post on GitLab individual AC Project.
- Finalized Challenge Programs (EL team in coordination with AC teams and DARPA): Final release on 1/15/2016. Post on GitLab EL_and_AC_Teams Project.
- Engagement 1 (all STAC teams): Event runs from 2/16/2016-2/18/2016 with PI meeting to follow.
- Quick-Look Engagement 1 Results (EL team): Initial Engagement 1 Quick-Look Results released on 3/18/2016. Post on GitLab Public_EL_Information Project.
- Site Visits (All STAC teams): March 28th, 2016 through April 11th, 2016 at each of the team sites. Assume EL and DARPA (Julie Konnor) will attend.
- Final Engagement 1 Analysis Results (EL team): Full Engagement 1 Analysis Results released on 5/15/2016. Post on GitLab Public_EL_Information Project.

4.1.1.4 Engagement 1 Logistics and Detailed Schedule

Engagement 1 will occur from Tuesday, February 16th, 2016 through Thursday, February 18th, 2016. The detailed schedule is:

- Tuesday, February 16th, 2016:
  - 8:00 AM: EL team arrives for engagement setup.
  - 11:00 AM: TA 1 and TA3 arrive for system setup.
  - 1:00 PM: Engagement starts with the distribution of the password for the stac_challenge_distribution.
  - 4:30PM: Day 1 engagement ends. Teams are welcome to leave processes running overnight, but no work should be done by the teams on any part of the problem (honor system).
- Wednesday, February 17th, 2016:
  - 8:30 AM: Engagement restarts for all teams.
  - 12:00 Noon: Everyone pauses engagement for lunch.
  - 1:00 PM: Engagement restarts for all teams.
  - 4:30PM: Day 2 engagement ends. Teams are welcome to leave processes running overnight, but no work should be done by the teams on any part of the problem (honor system).
- Thursday, February 18th, 2016:
  - 8:30 AM: Engagement restarts for all teams.
  - 12:00 Noon: Engagement finishes for TA 1 and TA 3.

- 1:00 PM: EL team arrives for engagement takedown.
- 3:00PM: EL team leaves. Engagement finished.

Each team will bring 3-5 people (if you want to bring more than 5 or fewer than 3, please request this to both DARPA and the EL team). Anyone who participates in any portion of the engagement will count as having participated in the entire day of the engagement (day 1 and 3 count as a half day). The EL team will record the total number of person days each team uses for the engagement and will normalize each team's performance by the number of person days involved.

4.1.1.5 Post Engagement Analysis Plans

4.1.1.5.1.1 Metrics

Each team's performance will be addressed with respect to the three metrics provided in the STAC BAA: Speed, Scale and Accuracy.

- Speed will be a measure of the rate at which the team analyzes challenge programs using their techniques and tools. This will be measured in terms of the number of challenge programs analyzed per person day (normalized by the number of days of the engagement and people on each team). We will also track the number of instructions analyzed per person day.
- Accuracy will be a measure of the proportion of challenge program questions that are correctly identified as containing or not containing a space or time vulnerability. The number of vulnerabilities correctly found (true positives) and the number of vulnerabilities declared when no true vulnerability exists (false positives) will be tracked.
- Scale will be a measure of the size of the challenge program that the teams can handle. For engagement 1 this is expected to be measured in terms of instruction counts though other measures of scale also will be considered.

4.1.1.5.1.2 Initial Assessments, Refinements Based on Blue Team Feedback and Site Visits

Immediately after the engagement, the EL team will provide an initial assessment of the performance of each team against the program metrics using the intended vulnerabilities (i.e., those vulnerabilities that were intentionally added to the challenge programs). This information will not be provided to the R&D Teams as the engagement 1 challenge problems will be used for the engagement 2 take home event. Aggregated forms of this information that minimize side channels on individual team performance will be posted as possible.

4.1.2 Engagement Results

4.1.2.1 Overview

In Engagement 1 (E1), 42 questions were asked about 19 different challenge programs. Of the 19 challenge problems, 14 were provided by CyberPoint and 5 were provided by Raytheon/BBN Technologies (BBN). The 42 questions can be broken into 9 different categories. 5 different question types were asked (Algorithmic Complexity (AC) in Time, AC in Space, Side Channel (SC) in Time, SC in Space, and SC in Space and Time) with two intended question results (positive or null) asked for four of the types and one intended question result (positive) asked for the final question type. Table 1 provides the breakdown of the 42 questions asked.

Table 3: How many questions were asked for a given question type and intended question result

Question Type	Intended Question Result	Number of Questions
Algorithmic Complexity in Time	Positive	13
Algorithmic Complexity in Space	Positive	2
Side Channel in Time	Positive	4
Side Channel in Space	Positive	3
Side Channel in Time and Space	Positive	1
Algorithmic Complexity in Time	Null	3
Algorithmic Complexity in Space	Null	6
Side Channel in Time	Null	6
Side Channel in Space	Null	4

Eight R&D Teams and one control team, collectively known as the Blue Teams, participated in E1. In total, 82 questions were attempted by the Blue Teams (with 65 of those being attempted by the R&D Team subset of the Blue Teams). The total combined accuracy of the Blue Teams was 64.6% (63.1% for just the R&D Teams alone). The Blue Teams tested their tools on challenge programs created by 2 adversarial teams, or Red Teams. Table 1-2 provides listing of the teams the participated in E1.

Of the 82 questions attempted, 54 of them were AC in Time questions with a total accuracy of 77.8% (77.3% for the researchers alone). The number of AC in Time questions answered compared to the number of AC in Time questions asked is disproportionately large (66% to 31% respectively) and suggests that both the researchers and the control team are currently most comfortable handling AC in Time questions.

The control team, as expected, answered the most question with 17 (with 12 correct answers). University of Maryland (UMD) answered 15 questions (with 11 correct answers) and was the top R&D Team.

Table 4: The different teams that participated in E1

Blue Teams	Research Teams	North Eastern University (NEU)
		University of Colorado Boulder (UC Boulder)
		GrammaTech
		Draper Labs
		University of Utah
		Iowa State University (ISU)
		University of Maryland (UMD)
		Vanderbilt University
	Control Team	Invincea
Red Teams		CyberPoint
		BBN

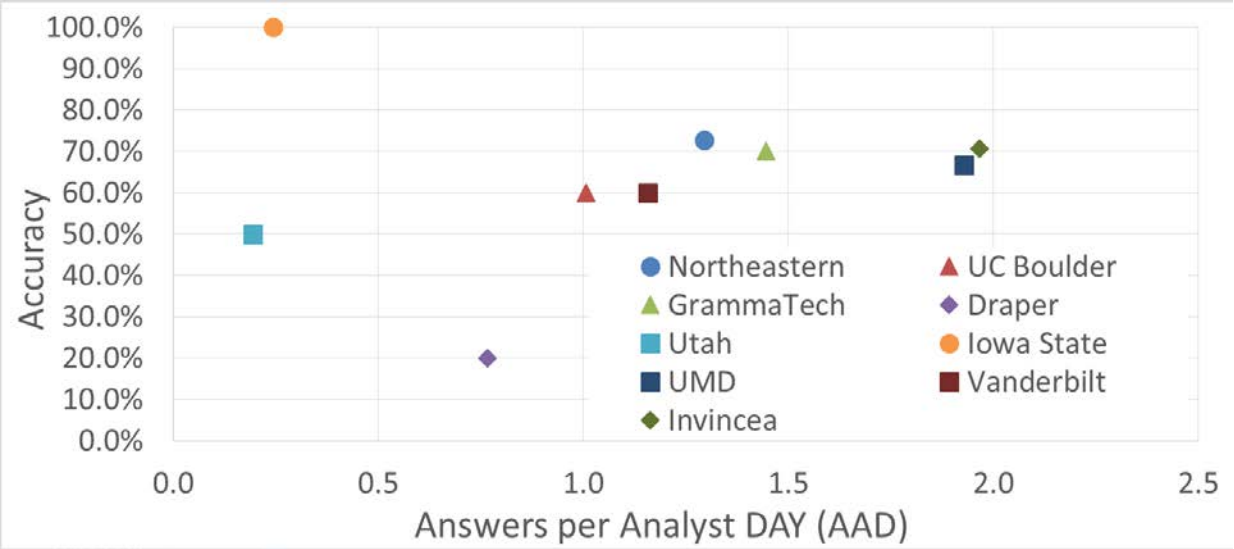


Figure 19:Answers per Analyst Day versus Initial Accuracy

Figure 19 provides the raw E1 results in the form of Answers per Analyst Day (AAD) versus Accuracy. AAD is calculated in the following way. The engagement is broken into half day sessions (4 in total). For each half day session, the number of people for each team is counted. Then, the number of team members is multiplied by the number of hours in the half day session and the total number of analyst hours is summed over all the engagement sessions for a given team. The number of analyst hours is then divided by eight (assuming a conventional eight-hour work day) to receive the total number of analyst days. The total number of questions answered (correct and incorrect) for each team is divided by the number of analyst days for that team to generate their AAD value. Thus, teams that answered more questions with fewer analysts would have a higher AAD score. Consequently, teams with many analysts that answered fewer questions will have a lower AAD score. The total combined AAD score for the Blue Teams was 1.09 (0.97 for the R&D Teams alone). If we weight the total accuracy according to AAD scores, we find the normalized total accuracy to be 65.3% (64.0% for the R&D Teams alone).

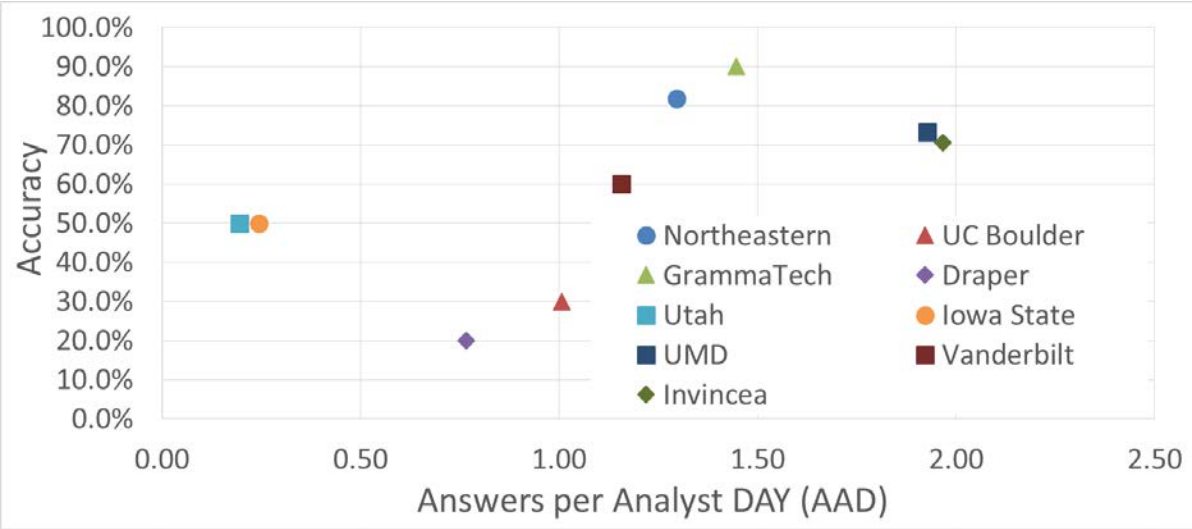


Figure 20:Answers per Analyst Day versus Corrected Accuracy

As is seen in Figure 20, Invincea and UMD have the highest AAD scores with decent accuracies. Iowa State has the best raw data accuracy (100%), but this is due to the fact that they only answered two questions and correctly answered both—which is reflected in their AAD score. The accuracies that are reported are likely to improve post engagement as we verify the existence of unintended vulnerabilities reported by the Blue Teams that may (or may not) exist in the challenge programs.

Figure 20 shows the corrected accuracy versus the performers' AAD scores. After post-engagement analysis on the Blue Teams' responses, various answers were changed from correct to incorrect or vice versa based on the validity of the responses. In particular, the TextCrunchr app was found to contain an unintended AC in time vulnerability dealing with highly compressed input. Those performers that found this vulnerability were given credit for the discovery (and many performers found this vulnerability). This is the cause of the majority of the change in accuracy seen between Figures 1-1 and 1-2. The analysis of the validity of the Blue Teams' responses is found in the subsequent section.

The following tables show which questions each team got correct. In these tables, a 1 indicates a correct answer, a -1 indicates an incorrect answer, and a 0 indicates the question was not attempted by the team. Also, LED in the program column is the abbreviation of the Law Enforcement Database challenge and was abbreviated for space. These tables indicate that the majority of correct answers is from the TextCrunchr family of programs from both the intended and unintended vulnerabilities (though substantially more from the unintended vulnerabilities).

Program	?#	Type	Vuln?	NEU	Ucol	GT	Draper	Utah	ISU	UMD	Vanderbilt	Invincea
Blogger	16	AC Space	N	1	-1	0	1	0	0	0	0	0
Blogger	40	AC Time	Y	1	1	1	-1	0	0	1	0	1
GabFeed_1	7	AC Time	Y	0	0	0	0	0	0	0	0	1
GabFeed_1	10	SC Time	Y	0	0	0	0	0	0	-1	0	1
GabFeed_1	29	SC Time	N	0	0	0	0	0	0	0	0	0
GabFeed_1	38	AC Space	N	0	-1	0	0	0	0	0	0	0
GabFeed_2	6	AC Time	Y	0	1	0	0	0	0	0	0	1
GabFeed_2	32	SC Time	N	0	0	0	0	0	0	-1	0	0
GabFeed_3	23	SC Time	Y	0	-1	0	0	0	0	0	0	0
GabFeed_3	36	AC Time	Y	0	-1	0	0	0	0	0	0	-1
GabFeed_3	39	SC T&S	Y	0	0	0	0	0	0	0	0	-1
GabFeed_4	3	AC Time	Y	0	0	0	0	0	0	0	0	0
GabFeed_5	1	SC Space	N	0	-1	0	0	0	0	0	0	0
GabFeed_5	4	SC Space	N	0	0	0	0	0	0	0	0	0
GabFeed_5	9	SC Time	N	0	0	0	0	0	0	0	0	0
GabFeed_5	14	AC Time	N	0	0	0	0	0	0	0	0	0
Graph Analyzer	13	AC Space	N	0	0	0	0	0	0	0	0	0
Graph Analyzer	20	AC Time	Y	0	0	0	-1	0	0	0	0	0
Graph Analyzer	34	AC Space	Y	0	0	0	0	0	0	-1	-1	0
Image Processor	5	AC Time	Y	-1	-1	1	-1	0	-1	1	0	0
Image Processor	30	AC Space	N	0	-1	0	-1	0	0	0	0	0
LED	19	AC Time	N	-1	0	0	0	-1	0	0	-1	-1
LED	26	SC Time	Y	0	0	0	0	0	0	1	-1	1
LED	27	SC Space	N	0	0	-1	0	0	0	0	-1	-1
SnapBuddy_1	12	SC Time	N	0	0	0	0	0	0	0	0	0
SnapBuddy_1	25	AC Space	Y	0	0	0	0	0	0	0	0	0
SnapBuddy_1	31	SC Space	Y	0	0	0	0	0	0	0	0	0
SnapBuddy_2	2	SC Space	N	0	0	0	0	0	0	0	0	0
SnapBuddy_2	15	SC Time	Y	0	0	0	0	0	0	-1	0	0
SnapBuddy_2	28	SC Space	Y	0	0	0	0	0	0	0	0	0
SnapBuddy_2	35	AC Space	N	0	0	0	0	0	0	0	0	0
SnapBuddy_3	18	SC Time	N	0	0	0	0	0	0	0	0	0
SnapBuddy_3	22	AC Time	Y	0	0	0	0	0	0	0	0	1
SubSpace	41	SC Time	N	0	0	0	0	0	0	0	0	-1
SubSpace	42	SC Space	Y	0	0	0	0	0	0	1	0	1
TextCrunchr_1	21	AC Time	Y	1	0	1	0	1	1	1	1	0
TextCrunchr_2	37	AC Time	Y	1	0	1	0	0	0	1	1	0
TextCrunchr_3	11	AC Time	Y	1	0	1	0	0	0	1	1	1
TextCrunchr_5	17	AC Space	N	1	0	1	0	0	0	1	0	1
TextCrunchr_5	24	AC Time	N	1	0	1	0	0	0	1	1	1
TextCrunchr_6	8	AC Time	Y	1	1	1	0	0	0	1	1	1
TextCrunchr_7	33	AC Time	Y	1	0	1	0	0	0	1	1	1

Figure 21: Correct (green), incorrect (red) and unanswered questions by performer

Table 3 provides the difference between accuracies when the TextCrunchr family of programs is removed from consideration. As is seen, most of the teams' accuracies are reduced (Vanderbilt's the most significantly). However, Utah's accuracy actually increases due to the fact that of the two questions they answered, the one they answered incorrectly was in TextCrunchr.

Table 5: Team accuracies with and without the TextCrunchr family of programs

Blue Team	Accuracy With TextCrunchr	Accuracy w/o TextCrunchr
Northeastern	81.8%	50%
UC Boulder	30.0%	22.2%
GrammaTech	90.0%	66.7%
Draper	20.0%	20.0%
Utah	50.0%	100%
Iowa State	50.0%	0.0%
UMD	73.3%	42.8%
Vanderbilt	60.0%	0.0%
Invincea	70.6%	58.3%

4.1.2.2 AC Teams

4.1.2.2.1 CyberPoint

CyberPoint produced 14 challenge programs with 30 questions attributed to the different challenge programs. The following table breaks down the number of intended positive and null questions for each of CyberPoint's challenge programs.

Table 6: Engagement 1 CyberPoint Challenge Performance

Challenge Program	# of Intended Positive Qs	# of Intended Null Qs	Total # of Qs	# of Attempts	# of Correct Attempts*
GabFeed_1	2	2	4	4	2
GabFeed_2	1	1	2	3	2
GabFeed_3	3	0	3	4	0
GabFeed_4	1	0	1	0	0
GabFeed_5	0	4	4	1	0
SnapBuddy_1	2	1	3	0	0
SnapBuddy_2	2	2	4	1	0
SnapBuddy_3	1	1	2	1	1
TextCrunchr_1	1	0	1	6	6
TextCrunchr_2	1	0	1	4	4
TextCrunchr_3	1	0	1	5	5
TextCrunchr_5	0	2	2	9	9
TextCrunchr_6	1	0	1	6	6
TextCrunchr_7	1	0	1	5	5

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

Overall, the Blue Teams answered 49 CyberPoint questions with 41 of them being correct (this results in an overall accuracy of 83.7%). A majority (35 in total, 35 correct) of these questions come from the textcrunchr apps. An unintended vulnerability within the textcrunchr apps related to the compression ratio of an input file is the reason for the 100% accuracy rate on the TextCrunchr app questions.

4.1.2.2.2 Raytheon/BBN Technologies

BBN produced 5 challenge programs with 12 questions attributed to the different challenge programs. The following table breaks down the questions asked between intended positive and null questions.

Table 7: Engagement 1 Raytheon/BBN Technologies Challenge Performance

Challenge Program	# of Intended Positive Qs	# of Intended Null Qs	Total # of Qs	# of Attempts	# of Correct Attempts*
Blogger	1	1	2	9	7
Graph Analyzer	2	1	3	3	0
Image Processor	1	1	2	8	2
Law Enforcement Database	1	2	3	10	2
Subspace	1	1	2	3	2

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

In total, 33 BBN questions were answered (13 of which were correctly answered) with an accuracy of 39.4%. Of BBN's challenge programs, Law Enforcement Database had the most questions answered, a total of 10 (1 correct) with Blogger following closely with 9 answers (7 correct).

4.1.2.2.2.1 Notes on the Image Processor Challenge

The intended vulnerability in Image Processor challenge program is in the 'filter' method of the Intensify class. This method takes an argument of a java.awt.image.BufferedImage object and iterates through the image pixel by pixel. For each pixel the challenge program calls the Mathematics.intensify method 3 times using the RGB values of the current pixel and 6 neighboring pixels to the left and right of the current pixel. The Mathematics.intensify method calls the Mathematics.exp method three times for each RGB value. The Mathematics.exp function takes arguments of a base and an exponent. The method sequentially calculates the exponential value by iterating n times, where n is the provided exponent value. The exponent value provided to the Mathematics.exp method is calculated based on the provided RGB values for the Mathematics.intensify method. The exponent value is calculated by adding the provided RGB values and calculating the modulus 255 of the sum. The modulus value is used to call the 'accuracy' function which returns the exponent value. The 'accuracy' function is:

$$accuracy[x] = 4 + \frac{1}{\tan\frac{|30 - x|}{255} + 0.001}$$

Equation 1: RGB Accuracy Function

The function has a domain of 0 to 256 and a range of 4.8145 to 1004. The function reaches its maximum value at x=30 and is symmetric about x=30. The image processing challenge has a runtime of O (n * a) where n is the total number of pixels in the provided image and 'a' is the output of the accuracy function for the mean RGB value in the image. Given a fixed image of 'n' pixels the best case runtime of the challenge program is if the mean sum of the RGB values of the pixels is 254, 590, or 764. The worst case runtime is if the mean sum of the RGB values of the pixels 30, 285, or 540. The current budget for algorithmic complexity was set using a 250,000 pixel image with an accuracy function output of 1004. This results in 251 million operations for the challenge program. It is possible to exceed this number of operations and stay within the input budget by providing a highly compressible images with a large number of pixels. Given that the smallest output from the 'accuracy' function is 4, an image with 62.75 million pixels can cause the challenge program to run more operations than it did for the image which was used to set the operations budget for the challenge program.

There were two other unintended vulnerabilities within the Image Processor challenge program that would enable the user to exceed the resource usage limit and stay within the input budget. The first is the challenge program stores the paths to training images in a file which the user can modify, users can edit this file to apply multiple tags to the same training image. This would enable a user to exceed the resource usage limit. The second unintended vulnerability is that the challenge program prepends a fixed file path to all input image in order to enforce that users do cannot train images not in the provided training set. Users can edit the file used to store the paths to training images. Users can create paths from the prepended path added by the challenge program to the location of any file. This would allow users to add more images to the training set and exceed the resource usage limit.

4.1.2.2.2.2 Notes on the Law Enforcement Database Challenge

The Law Enforcement Database challenge is a database of private and public keys. Users can search a range of public keys and insert public keys. When the server receives a search request the server queries the data structure of the challenge program and receives a list of all keys (public and private) in the search range. The server processes the list of keys. If a key is public the server returns a packet containing the key. If a key is private the server throws an exception and performs other operations, but the private key is not returned. There are two side channels in the challenge program. The strong side channel is the timing difference between two public keys. The weak side channel is the timing difference between the first and last packets received by a user in response to a search request.

In the case of the strong side channel an attacker can query the entire range of public keys a number of times (N) and identify the distribution of response times for each gap between two public keys. The attacker can identify a threshold point from this information and identify which gap exceeds this threshold point. Using a combination of insert and search operations an attacker can run a binary search on the gap and identify the private key. Using this method, the attacker can identify the private key 99.9 % of the time using 40+N operations. The tested proof uses 100 samples resulting in 140 total operations.

In the case of the weak side channel an attacker can employ the same strategy described above to run a binary search on the full range of the database (40 million keys). This method would allow an attacker to identify the private key 38.4 % of the time using 46+8N operations. With the number of samples (N) at 50, an attacker would require 446 operations. While the number of operations satisfied the operational budget of 500 operations, the probability of success is well below the 95% probability of success in the operational budget.

The issue with the weak side channel is that given a search request, the standard deviations of short (no private key) and long (private key) response times will increase as the number of public keys in the search request increases. Given the operational budget of 500 operations, an attacker can sample two gaps N times and determine a threshold value from this. The attacker can then sample each gap separately by querying the server for the associated public keys that define the gap. Once the attacker identifies the gap containing the secret, the attacker can proceed to run a binary search on the range and identify the secret within 329 + N operations. The attacker will have a 98.3 % likelihood of correctly identifying the secret. With $N < 172$ the attacker can stay within the operational budget of 500 operations.

Given the operational budget of 500 operations it is possible for the Blue Teams to only identify the weaker side channel and design an exploit within the operational budget. This exploit is equivalent in design to the strong side channel in the challenge program.

4.1.2.3 Conclusions

There were three challenge problems in Engagement 1 that teams seem to have particularly struggled with: Image Processor (AC Time), Graph Analyzer (AC Time, AC Space), and the Law Enforcement Database (SC Time).

The Image Processor challenge program contained a complexity of O (n*m*p) (n is the number of tagged images plus the classification image; m is the number of pixels in each image; p is the output of a function based on RGB values of each pixel). Four Blue Teams responded incorrectly to this challenge program. One of the Blue Teams failed to identify the complexity within the challenge program; three teams failed to identify the full O(n*m*p) complexity path within the

challenge program; three teams failed to correctly determine the complexity bounds on the n, m, or p variables in the O(n*m*p) complexity of the challenge program. We believe that having challenge programs with vulnerabilities formed by taking the product of many different dimensions are likely to continue to cause problems for the current Blue Team approaches.

The Graph Analyzer challenge program contained a complexity of

$$(v+e)\left(\frac{v^2-v}{2}+v+e\right)$$

Equation 2: Graphic Analyzer Complexity

where v is the number of vertices and e is the number of edges in an input graph. The AC Time vulnerability within the challenge program is that in the 'fast' case, the user input (bytes) is linearly proportional to the number of vertices in the defined given graph ($v \propto Input\ Bytes$). In the 'slow' case the user input is exponentially proportional to the number of vertices in the defined graph ($v \propto e^{Input\ Bytes}$). The user can use the "container:" key word in their input file to trigger the 'slow' case of the challenge program. The AC time complexity bounds is not exceeded if the user selects the Post Script (PS) output format; it is only exceeded if a user selects a PNG output format. The AC space vulnerability is dependent on the output format of the challenge program as well. The size of a PNG file is limited to 16 MB (2000x2000 pixels, 32 bits per pixel) by the challenge program. The size of a PS output file is not limited by the challenge program. The AC space complexity bounds is not exceeded if the user specifies a PNG output format; it is only exceeded if a user selects the PS output format. In the Graph Analyzer challenge program the same user input graph can be used to trigger either an AC Time or AC Space vulnerability based on the specified output file. Two Blue Teams attempted this challenge program. Both teams only identified the 'fast' case of the challenge program and were therefore able to bind the challenge program such that there was no complexity in either time or space. We believe that programs that have a vulnerability that is linear with respect to the local data structure, but that the local data structure is super linear with respect to the user input will continue to cause problems for the current Blue Team approaches.

The Law Enforcement Database challenge program contained two SC Time vulnerabilities (one weak SC, one strong SC). When the server responds to a search request the resultant public keys in the search range are provided one packet per public key. The strong side channel exists in the timing difference between two consecutive public keys that contain a private key between them (e.g. 500, 650, and 700 where 650 is private and 500 and 700 are public) and two consecutive public keys that do not contain a private key between them (e.g. just 500 and 700). The weak side channel exists in the total time difference between when the first and last packet are received from the server for a requested search range. To be fair, these are really expressions of the same side channel. However, when the requested range includes many public keys, the compounded noise present in the total response time can obscure the 'skipped heart beat' of the private key. The operational budget is only sufficient using the strong version of the side channel. Three Blue Teams attempted this question. One Blue Team misread the question, one Blue Team did not provide strong justification for their response, and the last Blue Team identified the weak side

channel. We believe challenge programs with side effects that create differential side channels will continue to cause problems for the current Blue Team approaches.

4.2 Engagement 2

4.2.1 Engagement Plan

4.2.1.1 Purpose and Program Scope

4.2.1.1.1 Program Scope

This Engagement Plan is for Engagement 2 which is a take-home engagement. All engagement answers must be submitted by noon Eastern time on Monday July 25th, 2016. Answers should be sent to evan.fortunato@apogee-research.com or uploaded to Apogee's Gitlab server.

The document is structured into the following sections.

Section 1: Purpose and Program Scope
Section 2: Operational Definitions and Research Hypotheses
Section 3: Engagement 2 Plans and Logistics
Section 4: Post-Engagement Analysis Plans

4.2.1.2 Operational Definitions and Program Hypotheses

4.2.1.2.1 Operational Definitions of STAC Vulnerabilities

As a means to guide the Engagements, the STAC AC, EL, and government team has come up with operational definitions of vulnerability to algorithmic complexity attack and to side channels. The AC performers will use these definitions to determine when the vulnerabilities they've inserted into their challenge programs are severe enough for use in engagements. The Control and R&D Teams will use these operational definitions to determine when they've found a vulnerability worth reporting – i.e., if it meets the definition of a STAC vulnerability. [The operational definition document specifies the definitions for STAC vulnerabilities along with guidance for Blue Team responses to engagement challenges.]

4.2.1.2.1.1 Unintended Vulnerabilities in Challenge Programs

Many Algorithmic Complexity and Side Channel vulnerabilities may exist in each challenge program beyond those intentionally inserted. For example, a normal authentication function is technically a side channel that leaks a secret. However, most of these vulnerabilities are "weak" – e.g., for Algorithmic Complexity, the additional complexity is a small percentage of the normal complexity, and for Side Channels, the number of operations necessary to resolve the secret is impractically large.

The defined operational definitions defined above are designed to minimize these unintended vulnerabilities in the challenge programs with the use of strong vulnerability thresholds (e.g., adversary operation budget is small for Side Channels). These thresholds will serve to discount most of these "unintended" vulnerabilities as not meeting the STAC definition. However, some unintended vulnerabilities will occur that do meet the definition. A reported actual vulnerability that satisfies the operational definition criterion for a given challenge program is a correct

response, regardless if it is an intended or unintended vulnerability on behalf of the AC teams. During the analysis period after each Engagement, the EL team will work with the R&D and Control performers to verify that reported unintended vulnerabilities do actually meet the operational definition so that full credit is given.

4.2.1.2.2 Research Hypothesis and Segmentation of Challenge Problems

Throughout the STAC program, the EL team will have a set of research hypotheses pertaining to performance of the tools with respect to different types of STAC vulnerabilities and challenge problems. Example hypotheses include:

4. Complexity vulnerabilities can be due to: (1) a change in the complexity of the algorithm, (2) a change in the coefficients of a fixed complexity, or (3) a flawed data guard that allows processing of input data that shouldn't be allowed. How do the different R&D techniques perform with respect to each of these types of complexity vulnerabilities?
5. Side channels can be weakened by increasing the number of secret symbols that are mapped to identical observed symbols or by increasing the width of the parent distributions of the observed symbols (causing overlaps of the of the samples from different observed symbol distributions). How do the different R&D techniques perform with respect to each of these challenges?
6. Some non-vulnerable programs are locally balanced (e.g., all paths through a sub-section of the binary consume equivalent time and space resources) while others are non-locally balanced (e.g., resource consumption is asymmetric on a set of paths in multiple parts of the program resulting in entire execution traces that are balanced). How do the different R&D techniques perform with respect to the locality of balancing of resource consumption?

In support of these hypotheses, each challenge problem will be assessed and categorized with respect to how results on the challenge problem will support resolution of each of the hypotheses. This segmentation of the challenge problems will not be exposed during the engagements, but will be tracked for post-engagement analysis.

During the engagements, each challenge problem will come questions such as:

5) Does the challenge problem contain a time-based side channel?
6) Does the challenge problem contain a space-based side channel?
7) Does the challenge problem contain a time-based complexity vulnerability?
8) Does the challenge problem contain a space-based complexity vulnerability?

Note, that the questions will be more specific version of these classes of questions including additional context information and could include questions regarding combined time and space vulnerabilities.

4.2.1.3 Engagement 2 Plans and Logistics

4.2.1.3.1 Engagement 2 Schedule

The schedule below provides an overview of the milestones to happen during the six months of Engagement 2, including the take-home engagement itself, the site visits for post Engagement 1 analysis, and the post engagement evaluation.

- Engagement 2 (all STAC teams): Event runs from 2/18/2016-7/25/2016.
- Obtain and Release Updated Challenge Programs (AC and EL Teams): The AC teams will provide directed updates to the challenge programs post Engagement 1 to the EL team by 3/4/2016 and the EL team will release these updates to the Blue Teams by 3/15/2015.
- Release Engagement Plans (EL team): Initial release on 4/25/2016 with approximately monthly updates. Post on GitLab Public_EL_Information Project.
- Site Visits Post Engagement 1 (All STAC teams): 3/28/2016 through 4/11/2016 at each of the team sites.
- Quick-Look Engagement 2 Results (EL team): Initial Engagement 2 Quick-Look Results released on at the August PI meeting. Results will subsequently be posted on GitLab Public_EL_Information Project.
- Snapshot of R&D Team tools (R&D Teams): Research teams will deliver a snapshot of their tools to the EL team, the AC teams, and DARPA post Engagement 2 on 8/16/2016.
- Final Engagement 2 Analysis Results (EL team): Full Engagement 2 Analysis Results released on 10/25/2016. Post on GitLab Public_EL_Information Project.

4.2.1.4 Post Engagement Analysis Plans

4.2.1.4.1.1 Metrics

Each team's performance will be addressed with respect to the three metrics provided in the STAC BAA: Speed, Scale and Accuracy.

- Speed will be a measure of the number of questions that team answers during the engagement. Unlike live engagements, this will not be normalized by the number of analyst hours as there are no effective ways to track this number.
- Accuracy will be a measure of the proportion of challenge program questions that are correctly identified as containing or not containing a space or time vulnerability. The number of vulnerabilities correctly found (true positives) and the number of vulnerabilities declared when no true vulnerability exists (false positives) will be tracked.
- Scale will be a measure of the size of the challenge program that the teams can handle. For engagement 2 this is expected to be measured in terms of instruction counts though other measures of scale also will be considered.

4.2.1.4.1.2 Initial Assessments, Refinements Based on Blue Team Feedback and Site Visits

At the August PI meeting, the EL team will provide an initial assessment of the performance of each team against the program metrics using the intended vulnerabilities (i.e., those vulnerabilities that were intentionally added to the challenge programs).

4.2.2 Engagement Results

4.2.2.1 Overview

In Engagement 2, 42 questions were asked about 19 different challenge programs. Of the 19 challenge problems, 14 were provided by CyberPoint and 5 were provided by Raytheon/BBN Technologies (BBN). The 42 questions can be broken into 9 different categories. 5 different question types were asked (Algorithmic Complexity (AC) in Time, AC in Space, Side Channel (SC) in Time, SC in Space, and SC in Space and Time) with two intended question results (positive or null) asked for four of the types and one intended question result (positive) asked for the final question type. Table 8 provides the breakdown of the 42 questions asked.

Table 8: How many questions were asked for a given question type and intended question result.

Question Type	Intended Question Result	Number of Questions
Algorithmic Complexity in Time	Positive	13
Algorithmic Complexity in Space	Positive	2
Side Channel in Time	Positive	4
Side Channel in Space	Positive	3
Side Channel in Time and Space	Positive	1
Algorithmic Complexity in Time	Null	3
Algorithmic Complexity in Space	Null	6
Side Channel in Time	Null	6
Side Channel in Space	Null	4

Eight R&D Teams and one control team, collectively known as the Blue Teams, participated in the engagement. In total, 175 questions were attempted by the Blue Teams (with 133 of those being attempted by the R&D Team subset of the Blue Teams). The total combined accuracy of the Blue Teams was 77% (78% for just the R&D Teams alone). Table 9 provides a listing of the teams that participated in Engagement 2.

Table 9: The different teams that participated in Engagement 2

Blue Teams	Research Teams	University of Colorado Boulder (UC Boulder)
		Draper Labs
		GrammaTech
		Iowa State University (ISU)
		Northeastern University (NEU)
		University of Maryland (UMD)
		University of Utah
		Vanderbilt University
	Control Team	Invincea
Red Teams		CyberPoint
		BBN

A breakdown of the number of questions answered for each category and the resulting accuracy following an initial review by the EL is shown in Table 10.

Table 10: The distribution of Blue Team responses across different categories

	Total Responses	Correct Reviewed Responses	Accuracy
SC Space	23	21	91%
SC Time	42	28	67%
SC Space/Time	5	3	60%
AC Space	29	18	62%
AC Time	76	65	86%

Of the 175 questions attempted, 76 of them were AC Time questions with a total accuracy of 86% (88% for the R&D Teams). Blue teams performed well on SC Space vulnerabilities with 23 responses and a 91% accuracy (94% for the R&D Teams).

Engagement 2 contained a distribution of questions that were intended vulnerable and intended non-vulnerable. During the course of the engagement, Blue Teams discovered unintended vulnerabilities within the challenge programs. These unintended vulnerabilities were classified as "in-scope" if they satisfied the STAC operational definition requirements, and "out-of-scope" if they did not. "In-scope" vulnerabilities verified by the EL led to changes in the classification of vulnerable and non-vulnerable questions for all Blue Teams. "Out-of-scope" vulnerabilities led to changes in the classification of vulnerable and non-vulnerable questions only for the Blue Team that identified the vulnerability. The analysis of the Engagement 2 responses used this change in classification to evaluate the responses provided by the Blue Team.

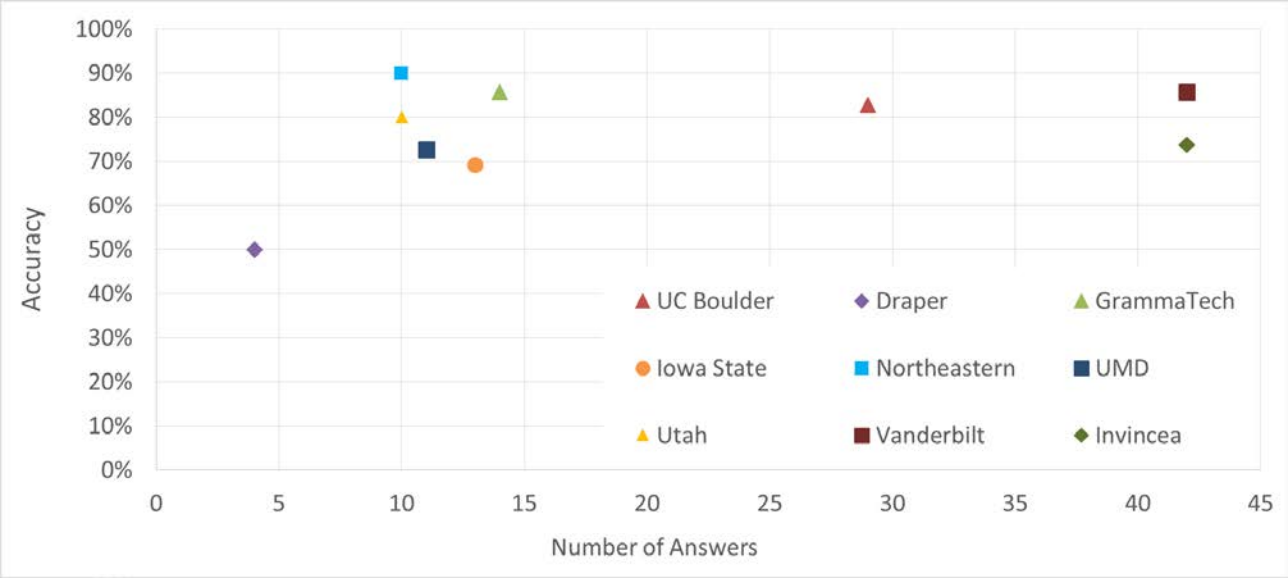


Figure 22: Number of answers versus Initial Accuracy

The initial analysis of the Engagement 2 responses used the raw responses provided by the Blue Teams. This analysis did not account for justifications. Figure 22 provides the raw Engagement 2 results in the form of number answers provided versus overall accuracy.

Following this initial analysis, the EL reviewed the justifications provided by the Blue Teams to determine if sufficient justification for an identified vulnerability or lack of vulnerability was provided. Figure 23 provides the reviewed Engagement 2 results

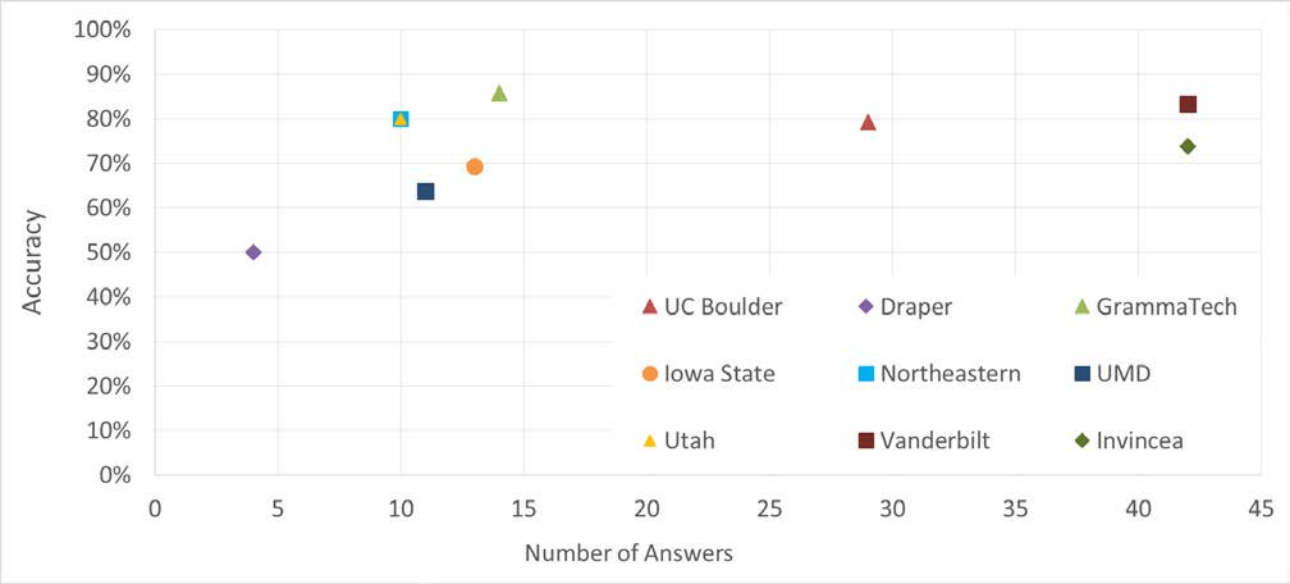


Figure 23: Number of Answers versus Reviewed Accuracy

As seen in Figure 23, UC Boulder, Vanderbilt, and Invincea outperformed the other Blue Teams. Vanderbilt had the second highest accuracy of 83% and responded to all 42 questions. UC Boulder had an accuracy of 79% and responded to 29 questions. Invincea had an accuracy of 74% and responded to all 42 questions. GrammaTech had the highest accuracy of 86% and responded to 14 questions.

The accuracy of positive responses by Blue Teams was 3% less than the accuracy of positive responses by the control team. The accuracy of negative responses by Blue Teams was approximately 7.1% greater than the accuracy of negative responses by the control team.

Figure 23 shows the performance of the Blue Teams in terms of accuracy and number of responses provided. This plot does not distinguish between vulnerable and non-vulnerable response accuracy. Figure 24 shows the probability of detection (the ratio of the number of correct vulnerable responses to total number of vulnerable questions answered) versus the true negative rate (the ratio of the number of correct non-vulnerable responses to the total number of non-vulnerable questions answered). Figure 24 accounts for justification by using the data from the set of reviewed responses for each Blue Team.

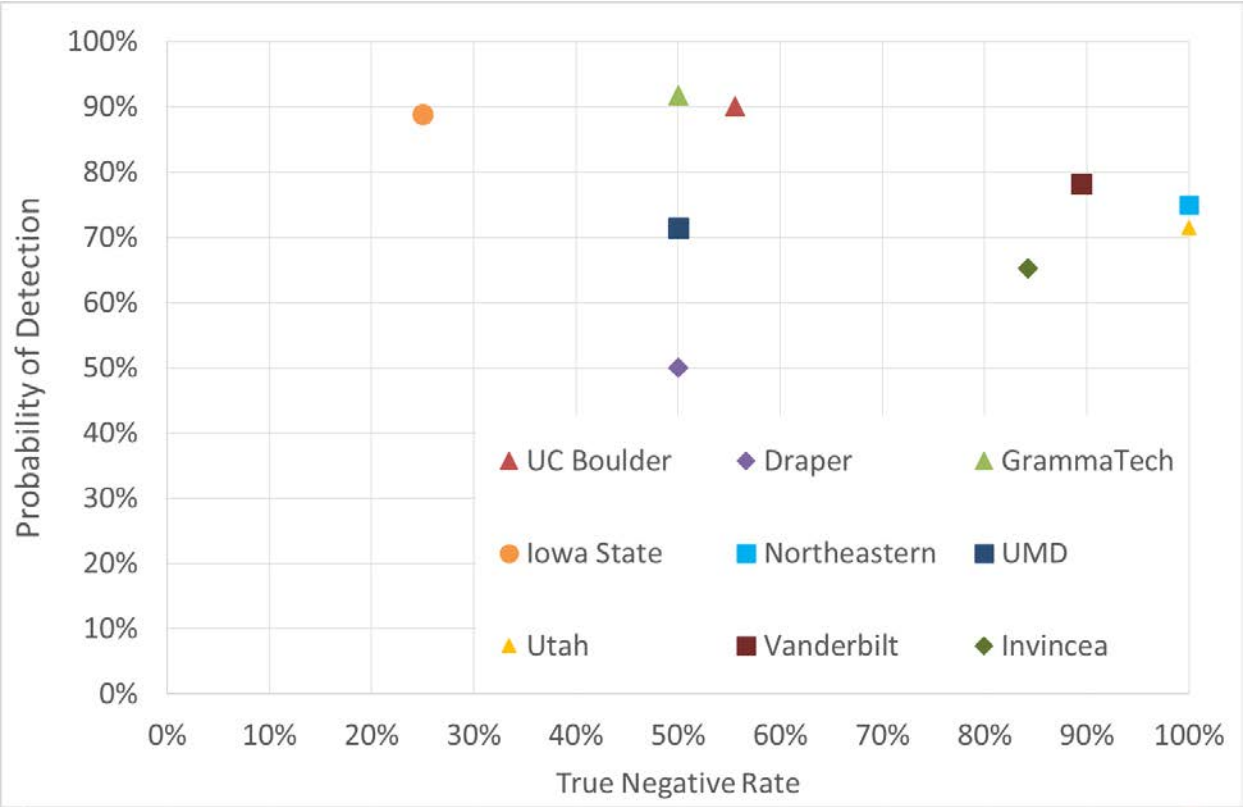


Figure 24: Probability of Detections vs True Negative Rate for Set of Answered Questions

From Figure 24, Northeastern and Utah had the highest true negative rate of 100% and GrammaTech has the highest probability of detection of 91.7% followed closely by UC Boulder with a probability of detection of 90%. Figure 24 distinguishes between vulnerable and non-vulnerable response accuracies and accounts for the justifications provided by the Blue Teams; however, it fails to account for the number of responses provided. Figure 1-4 captures this information by calculating probability of detection as the ratio of the number of correct vulnerable responses to total number of vulnerable questions and true negative rate as the ratio of the number of correct non-vulnerable responses to the total number of non-vulnerable questions. Figure 1-4 does not account for the justification provided by the Blue Teams, and is based on the total set of all questions provided for Engagement 2. Figure 25 shows that Vanderbilt had both the highest probability of detection and the highest true negative rate. UC Boulder had the second highest probability of detection and the third highest true negative rate, compared to Invincea which had the second highest true negative rate and the third highest probability of detection. Figure 24 and Figure 25 combine to showcase both the vulnerable and non-vulnerable accuracies for the Blue Teams and the ability of the Blue Teams to answer the total set of questions provided for Engagement 2.

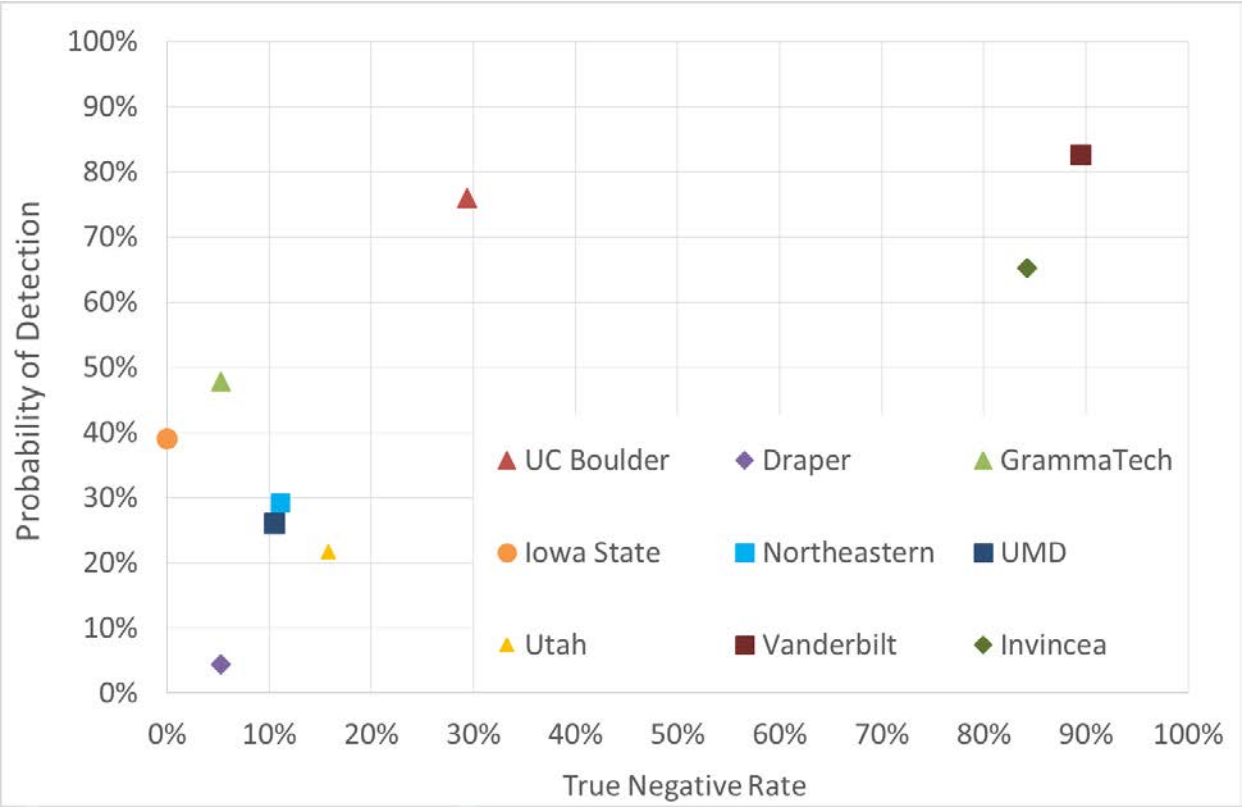


Figure 25: Probability of Detections vs True Negative Rate for Set of All Engagement 2 Questions

The following tables show which questions each team got correct. In these tables, a 1 indicates a correct answer, a -1 indicates an incorrect answer, and a 0 indicates the question was not attempted by the team. Also, LED in the program column is the abbreviation of the Law Enforcement Database challenge and was abbreviated for space.

Program	Question	Type	Vuln?	UC Boulder	Draper	GrammaTech	Iowa State
Blogger	16	AC Space	N	-1	1	0	-1
Blogger	40	AC Time	Y	1	-1	1	1
GabFeed_1	7	AC Time	Y	1	0	1	0
GabFeed_1	10	SC Time	Y	1	0	1	0
GabFeed_1	29	SC Time	N	0	0	0	0
GabFeed_1	38	AC Space	N	1	0	0	0
GabFeed_2	6	AC Time	Y	1	0	1	0
GabFeed_2	32	SC Time	N	1	0	0	0
GabFeed_3	23	SC Time	Y	1	0	0	1
GabFeed_3	36	AC Time	Y	1	0	0	0
GabFeed_3	39	SC T&S	Y	-1	0	0	1
GabFeed_4	3	AC Time	Y	1	0	1	0
GabFeed_5	1	SC Space	N	-1	0	0	1
GabFeed_5	4	SC Space	N	1	0	0	0
GabFeed_5	9	SC Time	N	1	0	0	0
GabFeed_5	14	AC Time	N	1	0	0	0
Graph Analyzer	13	AC Space	N	0	-1	0	0
Graph Analyzer	20	AC Time	Y	0	1	0	0
Graph Analyzer	34	AC Space	Y	-1	0	-1	0
Image Processor	5	AC Time	Y	1	0	1	0
Image Processor	30	AC Space	N	0	0	1	0
LED	19	AC Time	N	0	0	0	-1
LED	26	SC Time	Y	1	0	1	-1
LED	27	SC Space	N	1	0	0	0
SnapBuddy_1	12	SC Time	N	-1	0	0	-1
SnapBuddy_1	25	AC Space	Y	0	0	0	0
SnapBuddy_1	31	SC Space	Y	0	0	0	1
SnapBuddy_2	2	SC Space	N	0	0	0	0
SnapBuddy_2	15	SC Time	Y	1	0	0	0
SnapBuddy_2	28	SC Space	Y	1	0	0	0
SnapBuddy_2	35	AC Space	N	1	0	0	0
SnapBuddy_3	18	SC Time	N	0	0	0	0
SnapBuddy_3	22	AC Time	Y	1	0	0	1
SubSpace	41	SC Time	N	0	0	0	0
SubSpace	42	SC Space	Y	0	0	1	1
TextCrunchr_1	21	AC Time	Y	0	0	0	0
TextCrunchr_2	37	AC Time	Y	1	0	1	1
TextCrunchr_3	11	AC Time	Y	1	0	0	0
TextCrunchr_5	17	AC Space	N	-1	0	-1	0
TextCrunchr_5	24	AC Time	N	0	0	0	0
TextCrunchr_6	8	AC Time	Y	1	0	1	0
TextCrunchr_7	33	AC Time	Y	1	0	1	1

Figure 26: Correct (green), incorrect (red), and unaswered (yellow) questions from the engagement Part 1

Program	Question	Type	Vuln?	Northeastern	UMD	Utah	Vanderbilt	Invincea
Blogger	16	AC Space	N	0	0	0	1	1
Blogger	40	AC Time	Y	1	1	0	1	1
GabFeed_1	7	AC Time	Y	-1	0	0	1	1
GabFeed_1	10	SC Time	Y	-1	-1	0	-1	-1
GabFeed_1	29	SC Time	N	0	1	0	1	1
GabFeed_1	38	AC Space	N	0	0	0	1	1
GabFeed_2	6	AC Time	Y	0	0	0	1	1
GabFeed_2	32	SC Time	N	1	-1	0	1	1
GabFeed_3	23	SC Time	Y	0	0	0	1	1
GabFeed_3	36	AC Time	Y	0	-1	1	1	-1
GabFeed_3	39	SC T&S	Y	0	1	0	1	-1
GabFeed_4	3	AC Time	Y	0	0	0	1	-1
GabFeed_5	1	SC Space	N	0	0	0	1	1
GabFeed_5	4	SC Space	N	0	0	0	1	1
GabFeed_5	9	SC Time	N	1	0	0	1	-1
GabFeed_5	14	AC Time	N	0	0	0	1	1
Graph Analyzer	13	AC Space	N	0	0	0	1	1
Graph Analyzer	20	AC Time	Y	0	0	0	-1	1
Graph Analyzer	34	AC Space	Y	0	0	0	-1	1
Image Processor	5	AC Time	Y	1	1	1	1	1
Image Processor	30	AC Space	N	0	0	1	1	1
LED	19	AC Time	N	1	0	0	1	1
LED	26	SC Time	Y	0	1	0	1	1
LED	27	SC Space	N	0	0	0	1	-1
SnapBuddy_1	12	SC Time	N	0	0	0	1	1
SnapBuddy_1	25	AC Space	Y	0	0	0	-1	-1
SnapBuddy_1	31	SC Space	Y	0	0	0	1	1
SnapBuddy_2	2	SC Space	N	0	0	0	1	1
SnapBuddy_2	15	SC Time	Y	0	1	0	-1	-1
SnapBuddy_2	28	SC Space	Y	0	0	0	1	1
SnapBuddy_2	35	AC Space	N	0	0	0	1	1
SnapBuddy_3	18	SC Time	N	0	-1	0	-1	1
SnapBuddy_3	22	AC Time	Y	0	0	0	1	1
SubSpace	41	SC Time	N	0	1	0	-1	1
SubSpace	42	SC Space	Y	1	0	0	1	1
TextCrunchr_1	21	AC Time	Y	1	0	-1	1	-1
TextCrunchr_2	37	AC Time	Y	0	0	1	1	1
TextCrunchr_3	11	AC Time	Y	0	0	1	1	1
TextCrunchr_5	17	AC Space	N	0	0	1	1	-1
TextCrunchr_5	24	AC Time	N	0	0	1	1	1
TextCrunchr_6	8	AC Time	Y	0	0	1	1	1
TextCrunchr_7	33	AC Time	Y	1	0	-1	1	-1

Figure 27: Correct (green), incorrect (red), and unanswered (yellow) questions from the engagement Part 2

4.2.2.2 AC Teams

4.2.2.2.1 CyberPoint

CyberPoint produced 14 challenge programs with 30 questions attributed to the different challenge programs. Table 11 breaks down the number of intended positive and null questions for each of CyberPoint's challenge programs.

Table 11: Engagement 2 CyberPoint Programs and Questions

Challenge Program	# of Intended Positive Qs	# of Intended Null Qs	Total # of Qs	# of Attempts	# of Correct Attempts*	Percentage Correct
GabFeed_1	2	2	4	17	12	71%
GabFeed_2	1	1	2	9	8	89%
GabFeed_3	3	0	3	14	10	71%
GabFeed_4	1	0	1	4	3	75%
GabFeed_5	0	4	4	14	12	86%
SnapBuddy_1	2	1	3	9	5	56%
SnapBuddy_2	2	2	4	12	10	83%
SnapBuddy_3	1	1	2	7	5	71%
TextCrunchr_1	1	0	1	4	2	50%
TextCrunchr_2	1	0	1	6	6	100%
TextCrunchr_3	1	0	1	4	4	100%
TextCrunchr_5	0	2	2	8	5	63%
TextCrunchr_6	1	0	1	5	5	100%
TextCrunchr_7	1	0	1	7	5	71%
Total	17	13	30	120	92	77%

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

Overall, the Blue Teams answered 120 CyberPoint questions with 92 of them being correct (this results in an overall accuracy of 77%). Blue teams had an accuracy of 100% for TextCrunchr variants 2, 3, and 6 and struggled with SnapBuddy 1 and TextCrunchr 1 with accuracies of 56% and 50% respectively.

4.2.2.2.2 Raytheon/BBN Technologies

BBN produced 5 challenge programs with 12 questions attributed to the different challenge programs. Table 12 breaks down the questions asked between intended positive and null questions.

Table 12: Engagement 2 Raytheon/BBN Technologies Programs and Questions

Challenge Program	# of Intended Positive Qs	# of Intended Null Qs	Total # of Qs	# of Attempts	# of Correct Attempts*	Percentage Correct
Blogger	1	1	2	13	10	77%
Graph Analyzer	2	1	3	10	5	50%
Image Processor	1	1	2	11	11	100%
Law Enforcement Database	1	2	3	13	10	77%
Subspace	1	1	2	8	7	88%
Total	6	6	12	55	43	78%

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

In total, 55 BBN questions were answered (43 of which were correctly answered) with an accuracy of 78%. Of BBN's challenge programs, Blogger and Law Enforcement Database had the most questions answered, a total of 13 (77% accuracy for both). Blue teams struggled most with the Graph Analyzer challenge programs with 10 attempts and only a 50% accuracy.

4.3 Engagement 3

4.3.1 Engagement Plan

4.3.1.1 Purpose and Program Scope

4.3.1.1.1 Program Scope

This Engagement Plan is for Engagement 3 which is a live engagement. Engagement 3 will take place from January 23rd 2017 to January 25th 2017. Engagement answers must be submitted at the end of the live engagement on January 25th 2017. Answers should be sent to evan.fortunato@apogee-research.com or uploaded to Apogee's Gitlab server.

The document is structured into the following sections.

Section 1: Purpose and Program Scope
Section 2: Operational Definitions and Research Hypotheses
Section 3: Engagement 3 Plans and Logistics
Section 4: Post-Engagement Analysis Plans

4.3.1.2 Operational Definitions and Program Hypotheses

4.3.1.2.1 Operational Definitions of STAC Vulnerabilities

As a means to guide the Engagements, the STAC AC, EL, and government teams have come up with operational definitions of vulnerability to algorithmic complexity attack and to side channels. The AC performers will use these definitions to determine when the vulnerabilities

they've inserted into their challenge programs are strong enough for use in engagements. The Control and R&D Teams will use these operational definitions to determine when they've found a vulnerability worth reporting – i.e., if it meets the definition of a STAC vulnerability[The operational definition document specifies the definitions for STAC vulnerabilities along with guidance for Blue Team responses to engagement challenges.].

4.3.1.2.2 Research Hypothesis and Segmentation of Challenge Problems

Throughout the STAC program, the EL team will have a set of research hypotheses pertaining to performance of the tools with respect to different types of STAC vulnerabilities and challenge problems. Example hypotheses include:

7. Complexity vulnerabilities can be due to: (1) a change in the complexity of the algorithm, (2) a change in the coefficients of a fixed complexity, or (3) a flawed data guard that allows processing of input data that shouldn't be allowed. How do the different R&D techniques perform with respect to each of these types of complexity vulnerabilities?
8. Side channels can be weakened by increasing the number of secret symbols that are mapped to identical observed symbols or by increasing the width of the parent distributions of the observed symbols (causing overlaps of the of the samples from different observed symbol distributions). How do the different R&D techniques perform with respect to each of these challenges?
9. Some non-vulnerable programs are locally balanced (e.g., all paths through a sub-section of the binary consume equivalent time and space resources) while others are non-locally balanced (e.g., resource consumption is asymmetric on a set of paths in multiple parts of the program resulting in entire execution traces that are balanced). How do the different R&D techniques perform with respect to the locality of balancing of resource consumption?

In support of these hypotheses, each challenge problem will be assessed and categorized with respect to how results on the challenge problem will support resolution of each of the hypotheses. This segmentation of the challenge problems will not be exposed during the engagements, but will be tracked for post-engagement analysis.

During the engagements, each challenge problem will come questions such as:

9) Does the challenge problem contain a time-based side channel?
10) Does the challenge problem contain a space-based side channel?
11) Does the challenge problem contain a time-based complexity vulnerability?
12) Does the challenge problem contain a space-based complexity vulnerability?

Note, that the questions will be more specific version of these classes of questions including additional context information and could include questions regarding combined time and space vulnerabilities.

4.3.1.3 Engagement 3 Plans and Logistics

4.3.1.3.1.1 Engagement 3 Schedule

The schedule below provides an overview of the milestones to happen during the six months of Engagement 3, including the engagement itself and the post engagement evaluation.

- Engagement 3 (all STAC teams): Event runs from 1/23/2017- 1/25/2017.
- R&D teams will provide a snapshot of their toolset to DARPA, the AC, and EL teams by 09/19/2016.
- Release Engagement 3 Plan (EL team): 09/19/2016 by noon eastern time.
- AC Teams will deliver final versions of all Engagement 3 challenge programs to the EL: 11/21/2016 by noon eastern time.
- Engagement 3 (live engagement) will take place at DARPA from 1/23/2017 – 1/25/2017
- Quick-look Engagement 3 results (EL team): initial Engagement 3 quick-look results released at the PI meeting on 1/26/2017. Initial Engagement 3 analysis report (EL team) will be delivered to DARPA and the AC teams: 2/24/2017.
- Final Engagement 3 analysis report delivered to DARPA and the AC teams: 4/20/2017

4.3.1.4 Post Engagement Analysis Plans

4.3.1.4.1.1 Metrics

Each team's performance will be addressed with respect to the three metrics provided in the STAC BAA: Speed and Accuracy.

- Speed will be a measure of the number of questions that team answers during the engagement. The number of questions answered will be normalized by the number of analyst hours.
- Accuracy will be a measure of the proportion of challenge program questions that are correctly identified as containing or not containing a space or time vulnerability. The number of vulnerabilities correctly found (true positives) and the number of vulnerabilities declared when no true vulnerability exists (false positives) will be tracked. The number of true positives and false positives will be plotted against each other using true positive rate vs true negative rate plots to show the ability of the Blue Teams to accurately identify vulnerabilities and declare applications not vulnerable.

4.3.1.4.1.2 Initial Assessments, Refinements Based on Blue Team Feedback and Site Visits

At the January 26th PI meeting, the EL team will provide an initial assessment of the performance of each team against the program metrics using the intended vulnerabilities (i.e., those vulnerabilities that were intentionally added to the challenge programs).

4.3.2 Engagement Results

4.3.2.1 Overview

In Engagement 3, 57 questions were asked about 24 different challenge programs. Of the 24 challenge problems, 16 were provided by CyberPoint and 8 were provided by Raytheon/BBN Technologies (BBN). The 57 questions can be broken into 10 different categories. 5 different question types were asked (Algorithmic Complexity (AC) in Time, AC in Space, Side Channel (SC) in Time, SC in Space, and SC in Space and Time). Table 11 provides the breakdown of the 57 questions asked.

Table 13: How many questions were asked for a given question type and intended question result

Question Type	Intended Question Result	Number of Questions
Algorithmic Complexity in Time	Positive	11
Algorithmic Complexity in Space	Positive	10
Side Channel in Time	Positive	8
Side Channel in Space	Positive	4
Side Channel in Time and Space	Positive	1
Algorithmic Complexity in Time	Null	4
Algorithmic Complexity in Space	Null	5
Side Channel in Time	Null	6
Side Channel in Space	Null	6
Side Channel in Time and Space	Null	2

Eight R&D Teams and one control team, collectively known as the Blue Teams, participated in the engagement. In total, the Blue Teams attempted 117 questions (with 114 of those being attempted by the R&D Team subset of the Blue Teams). The total combined accuracy of the Blue Teams was 59% (59% for just the R&D Teams alone). Table 14 provides a listing of the teams that participated in Engagement 2.

Table 14:The different teams that participated in Engagement 2

Blue Teams	Research Teams	University of Colorado Boulder (UC Boulder)
		Draper Labs
		GrammaTech
		Iowa State University (ISU)
		Northeastern University (NEU)
		University of Maryland (UMD)
		University of Utah
		Vanderbilt University
	Control Team	Invincea
Red Teams		CyberPoint
		BBN

A breakdown of the number of questions answered for each category and the resulting accuracy following an initial review by the EL is shown in Table 15.

Table 15: The distribution of Blue Team responses across different categories

	Total Responses	Correct Reviewed Responses	Accuracy
SC Space	21	12	57%
SC Time	25	13	52%
SC Space/Time	3	0	0%
AC Space	31	20	65%
AC Time	37	24	65%

Of the 117 questions attempted, 37 of them were AC Time questions with a total accuracy of 65% (66% for the R&D Teams). Blue teams had the highest accuracy on AC questions with 68 responses and a 65% accuracy.

Engagement 3 contained a distribution of questions that were intended vulnerable and intended non-vulnerable. During the course of the engagement, Blue Teams discovered unintended vulnerabilities within the challenge programs. These unintended vulnerabilities were classified as "in-scope" if they satisfied the STAC operational definition requirements, and "out-of-scope" if they did not. "In-scope" vulnerabilities verified by the EL led to changes in the classification of vulnerable and non-vulnerable questions for all Blue Teams. "Out-of-scope" vulnerabilities led to changes in the classification of vulnerable and non-vulnerable questions only for the Blue Team that identified the vulnerability. The resulting plots of accuracy versus answers weighted per analyst day are shown below.

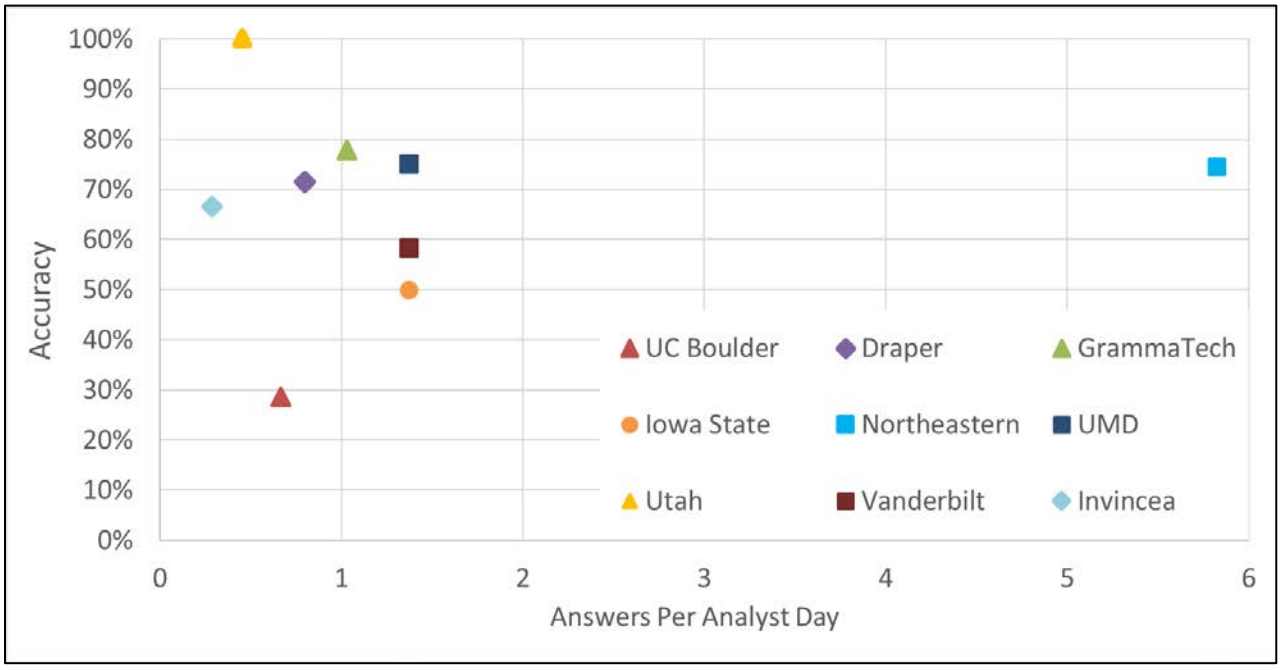


Figure 28: Number of answers versus Initial Accuracy

The initial analysis of the Engagement 3 responses used the raw responses provided by the Blue Teams. This analysis accounts for verified unintended vulnerabilities but not for justifications. Figure 28 provides the raw Engagement 3 results in the form of number answers provided versus overall accuracy.

Following this initial analysis, the EL reviewed the justifications provided by the Blue Teams to determine if sufficient justification for an identified vulnerability or lack of vulnerability was provided. Figure 29 provides the reviewed Engagement 3 results.

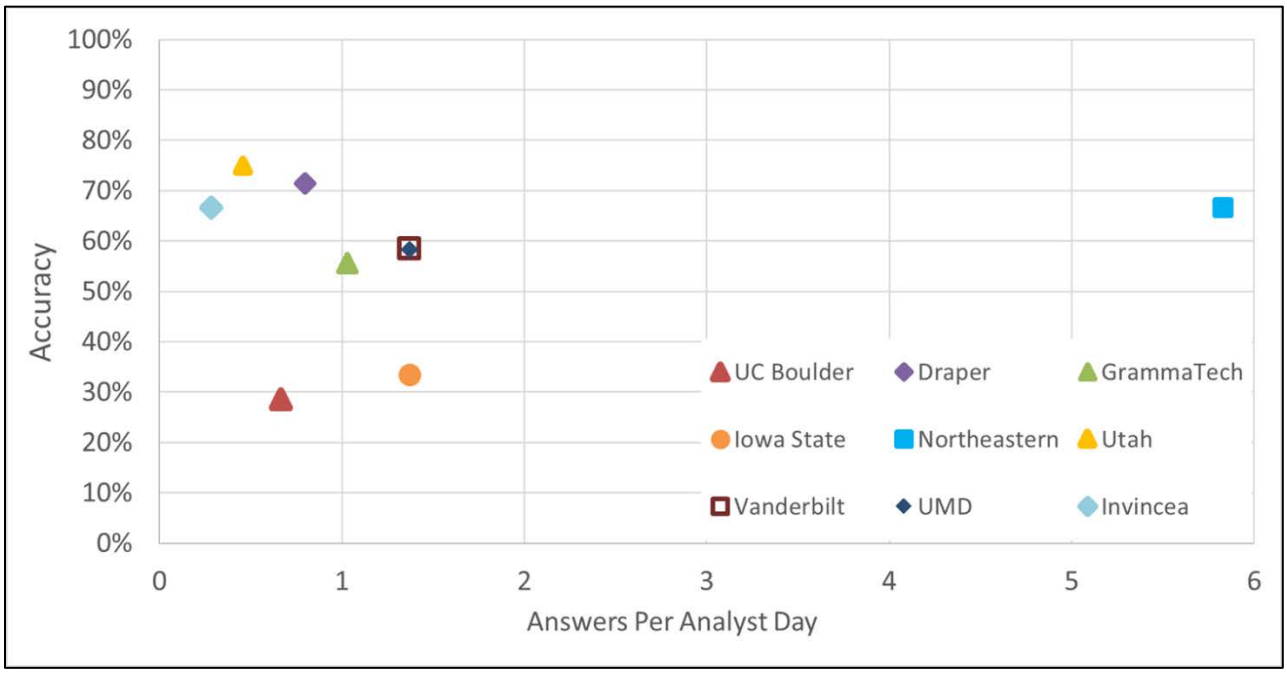


Figure 29: Number of Answers versus Reviewed Accuracy

As seen in Figure 29, Northeastern answered significantly more questions than any of the other Blue Teams and had an accuracy of 67%. Utah and Draper had the highest and second highest accuracy with 75% and 71% accuracies respectively.

The 59% accuracy of responses by Blue Teams was the same as the accuracy of the control team.

Figure 29 shows the performance of the Blue Teams in terms of accuracy and number of responses provided. This plot does not distinguish between vulnerable and non-vulnerable response accuracy. Figure 30 shows the true positive rate (the ratio of the number of correct vulnerable responses to total number of vulnerable questions answered) versus the true negative rate (the ratio of the number of correct non-vulnerable responses to the total number of non-vulnerable questions answered). Figure 30 accounts for justification by using the data from the set of reviewed responses for each Blue Team.

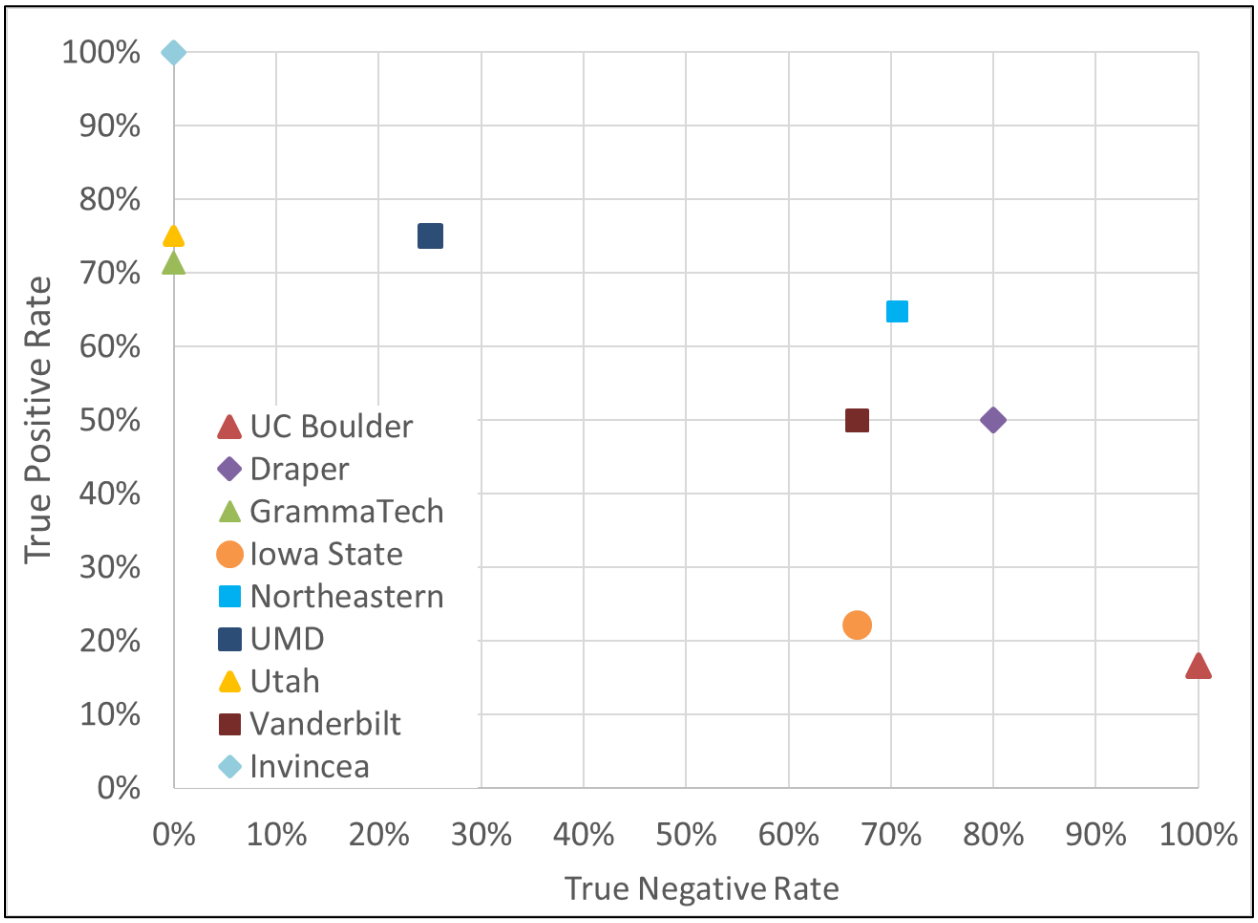


Figure 30: True Positive Rate vs True Negative Rate for Set of Answered Questions

From Figure 30, Invincea had the highest true positive rate of 100%. UC Boulder had the highest true negative rate of 100% but a true positive rate of 17 %. Northeastern was the only team with a true positive rate and a true negative rate both greater than 50%. Utah, GrammaTech, and Invincea all had true negative rates of 0%.

The following tables show which questions each team got correct. In these tables, a 1 indicates a correct answer, a -1 indicates an incorrect answer, and a 0 indicates the question was not attempted by the team. Also, LED in the program column is the abbreviation of the Law Enforcement Database challenge and was abbreviated for space.

Program	Question	Type	Vuln?	Northeastern	UC Boulder	GrammaTech	Draper
airplan_1	Question_046	AC Space	N	1	0	0	0
airplan_1	Question_052	AC Time	Y	1	0	1	0
airplan_2	Question_006	SC Space	Y	1	0	0	0
airplan_2	Question_020	AC Time	Y	1	0	-1	0
airplan_3	Question_013	SC Space	Y	1	0	0	0
airplan_3	Question_022	SC Space	N	-1	0	0	0
airplan_3	Question_025	AC Time	Y	1	0	-1	0
airplan_4	Question_009	AC Time	Y	1	0	0	0
airplan_4	Question_030	SC Space	N	1	0	0	0
airplan_4	Question_048	AC Space	Y	-1	0	0	0
bidpal_1	Question_014	SC Time	Y	-1	0	0	0
bidpal_1	Question_044	AC Time	N	1	0	0	0
bidpal_2	Question_037	SC Time	Y	-1	0	0	0
bidpal_2	Question_041	AC Time	Y	-1	0	0	0
collab	Question_018	AC Time	N	1	1	0	-1
collab	Question_028	SC Space	N	1	1	0	1
collab	Question_042	SC Time	N	-1	0	0	1
info_trader	Question_002	AC Time	Y	1	-1	0	0
info_trader	Question_004	AC Space	Y	-1	-1	0	0
linear_algebra_platform	Question_005	AC Space	Y	-1	0	1	0
linear_algebra_platform	Question_012	AC Time	Y	1	-1	0	0
malware_analyzer	Question_031	AC Time	N	1	0	0	1
malware_analyzer	Question_035	AC Space	Y	1	-1	0	-1
powerbroker_1	Question_036	AC Space	Y	1	0	0	0
powerbroker_1	Question_050	SC Time	Y	-1	0	0	0
powerbroker_2	Question_019	AC Space	Y	1	0	0	0
powerbroker_2	Question_056	SC Time	N	-1	0	0	0
powerbroker_3	Question_021	SC Time/Space	N	0	0	0	0
powerbroker_3	Question_023	SC Time/Space	N	0	0	0	0
powerbroker_3	Question_034	SC Time	Y	-1	0	0	0
powerbroker_4	Question_011	SC Time	Y	0	0	0	0
rsa_commander	Question_033	AC Time	Y	1	0	0	1
rsa_commander	Question_040	SC Time	N	-1	0	0	1
smartmail	Question_015	AC Time	N	1	0	0	0
smartmail	Question_024	AC Space	N	1	0	0	0
smartmail	Question_039	SC Time/Space	Y	-1	0	0	0
tour_planner	Question_003	SC Space	N	0	0	0	0
tour_planner	Question_008	AC Time	N	0	0	0	0
tour_planner	Question_038	SC Time	Y	0	0	0	0
tweeter	Question_017	AC Space	Y	1	0	0	0
tweeter	Question_049	AC Time	Y	-1	-1	0	0
withmi_1	Question_007	AC Space	Y	1	0	0	0
withmi_1	Question_010	SC Space	Y	-1	0	1	0
withmi_1	Question_053	SC Space	N	1	0	-1	0
withmi_2	Question_016	AC Space	Y	1	0	0	0
withmi_2	Question_029	AC Time	Y	1	0	0	0
withmi_3	Question_047	SC Time	N	1	0	0	0
withmi_3	Question_055	AC Space	Y	1	0	1	0
withmi_4	Question_027	SC Time	Y	1	0	0	0
withmi_4	Question_043	SC Time	Y	1	0	0	0
withmi_4	Question_045	AC Space	Y	-1	0	1	0
withmi_4	Question_051	SC Space	Y	1	0	0	0
withmi_5	Question_001	AC Space	Y	1	0	0	0
withmi_5	Question_026	SC Space	N	-1	0	-1	0
withmi_5	Question_032	SC Time	N	1	0	0	0
withmi_6	Question_054	AC Time	N	1	0	0	0
withmi_6	Question_057	AC Space	N	1	0	0	0

Figure 31: Correct (green), incorrect (red), and unanswered (yellow) responses from engagement Part 1

Program	Question	Type	Vuln?	Utah	Iowa State	UMD	Vanderbilt	Invincea
airplan_1	Question_046	AC Space	N	0	1	0	0	0
airplan_1	Question_052	AC Time	Y	0	-1	0	-1	0
airplan_2	Question_006	SC Space	Y	0	-1	0	0	0
airplan_2	Question_020	AC Time	Y	1	0	0	0	0
airplan_3	Question_013	SC Space	Y	0	0	0	0	0
airplan_3	Question_022	SC Space	N	0	0	0	0	0
airplan_3	Question_025	AC Time	Y	0	0	1	0	0
airplan_4	Question_009	AC Time	Y	0	0	0	0	0
airplan_4	Question_030	SC Space	N	0	0	-1	0	0
airplan_4	Question_048	AC Space	Y	0	0	0	0	0
bidpal_1	Question_014	SC Time	Y	0	-1	1	0	0
bidpal_1	Question_044	AC Time	N	0	0	0	0	0
bidpal_2	Question_037	SC Time	Y	0	0	-1	0	0
bidpal_2	Question_041	AC Time	Y	0	0	0	0	0
collab	Question_018	AC Time	N	0	0	0	1	0
collab	Question_028	SC Space	N	0	0	0	-1	0
collab	Question_042	SC Time	N	0	0	-1	1	0
info_trader	Question_002	AC Time	Y	1	1	0	0	0
info_trader	Question_004	AC Space	Y	-1	-1	0	0	0
linear_algebra_platform	Question_005	AC Space	Y	1	0	0	0	0
linear_algebra_platform	Question_012	AC Time	Y	0	-1	0	0	0
malware_analyzer	Question_031	AC Time	N	0	-1	0	0	0
malware_analyzer	Question_035	AC Space	Y	0	-1	0	0	1
powerbroker_1	Question_036	AC Space	Y	0	0	0	0	0
powerbroker_1	Question_050	SC Time	Y	0	0	0	0	0
powerbroker_2	Question_019	AC Space	Y	0	0	0	0	0
powerbroker_2	Question_056	SC Time	N	0	0	0	0	0
powerbroker_3	Question_021	SC Time/Space	N	0	0	0	0	0
powerbroker_3	Question_023	SC Time/Space	N	0	0	0	0	0
powerbroker_3	Question_034	SC Time	Y	0	0	1	0	0
powerbroker_4	Question_011	SC Time	Y	0	0	1	0	0
rsa_commander	Question_033	AC Time	Y	0	0	0	0	0
rsa_commander	Question_040	SC Time	N	0	0	-1	0	0
smartmail	Question_015	AC Time	N	0	0	0	0	0
smartmail	Question_024	AC Space	N	0	0	0	1	0
smartmail	Question_039	SC Time/Space	Y	0	-1	0	-1	0
tour_planner	Question_003	SC Space	N	0	0	0	-1	0
tour_planner	Question_008	AC Time	N	0	1	0	0	-1
tour_planner	Question_038	SC Time	Y	0	0	0	-1	0
tweeter	Question_017	AC Space	Y	0	1	0	0	0
tweeter	Question_049	AC Time	Y	0	0	0	0	1
withmi_1	Question_007	AC Space	Y	0	0	0	0	0
withmi_1	Question_010	SC Space	Y	0	0	0	1	0
withmi_1	Question_053	SC Space	N	0	0	0	1	0
withmi_2	Question_016	AC Space	Y	0	0	0	0	0
withmi_2	Question_029	AC Time	Y	0	0	0	0	0
withmi_3	Question_047	SC Time	N	0	0	1	0	0
withmi_3	Question_055	AC Space	Y	0	0	1	0	0
withmi_4	Question_027	SC Time	Y	0	0	0	1	0
withmi_4	Question_043	SC Time	Y	0	0	1	0	0
withmi_4	Question_045	AC Space	Y	0	0	-1	0	0
withmi_4	Question_051	SC Space	Y	0	0	0	1	0
withmi_5	Question_001	AC Space	Y	0	0	0	0	0
withmi_5	Question_026	SC Space	N	0	0	0	0	0
withmi_5	Question_032	SC Time	N	0	0	0	0	0
withmi_6	Question_054	AC Time	N	0	0	0	0	0
withmi_6	Question_057	AC Space	N	0	0	0	0	0

Figure 32: Correct (green), incorrect (red), and unanswered (yellow) responses from engagement Part 2

4.3.2.2 AC Teams

4.3.2.2.1 Vulnerabilities

The vulnerabilities in the STAC E3 challenge programs in addition to spanning space and time side channel and algorithmic complexity vulnerabilities contained components which can be mapped to real-world vulnerabilities. Table 16 presents general categorization of the E3 vulnerabilities and links to research papers, presentations, Common Vulnerabilities and Exposures (CVEs) and Common Weakness Enumerations (CWEs) that are comparable to the specified category.

Table 16: Mapping of engagement challenges to CVEs and CWEs

Challenge Program (s)	Description	Vuln	Link Type	Links
Collab	Timing attacks on database indexing algorithms	SC T	Research Paper (Direct)	https://www.usenix.org/legacy/event/woot07/tech/full_papers/futoransky/futoransky.pdf
			Research Paper (Direct)	https://pdfs.semanticscholar.org/f100/b668ff5de9909919da47a85ac6f5ee019f22.pdf
PowerBroker, BidPal, WithMi	Timing attacks against RSA implementations using Montgomery multiplication	SC T	CVE-2013-5915 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-5915
			CVE-2004-2682 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2682
			CVE-2003-0147 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0147
			Research Paper (Direct)	https://tls.mbed.org/public/WSchindler-RSA_Timing_Attack.pdf
			Research Paper (Direct)	https://www.uclouvain.be/crypto/services/download/publications.pdf.9256d17be6b4cdd4.70646633362e706466.pdf
InfoTrader	Tree traversal algorithm with broken guard to prevent cycle detection	AC S	Textbook (Related)	http://www.springer.com/cda/content/document/cda_downloaddocument/9783642381225-c2.pdf?SGWID=0-0-45-1402898-p175152334

InfoTrader, Linear Algebra Platform, AirPlan, RSA Commander, WithMi	Improper input validation resulting in a broken input guard	AC S/T	CWE-20 (Related)	https://cwe.mitre.org/data/definitions/20.html
			CVE-2017-3807 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-3807
			CVE-2010-3717 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3717
AirPlan, WithMi	Complexity vulnerabilities in compression implementations	AC S/T	CVE-2016-4630 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4630
			CVE-2014-2746 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2746
			CVE-2012-4447 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4447
Malware Analyzer, WithMi	Integer overflow vulnerability resulting in memory utilization	AC S	CVE-2016-1968 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1968
			CVE-2017-6308 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-6308
			CVE-2016-6207 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6207
			CVE-2017-5627 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5627
			CVE-2016-1933 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1933
Linear Algebra Platform	Improper guard on input results in large memory allocation	AC S	CVE-2014-9192 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9192
AirPlan	Ford-Fulkerson vulnerable implementation	AC T	Textbook (Exact)	http://www.cse.unt.edu/~tarau/teaching/AnAlgo/Ford%E2%80%93Fulkerson%20algorithm.pdf

	Dijkstra's Algorithm vulnerable with negative weights	AC T	Textbook (Exact)	http://www.eecs.tufts.edu/~ccousi01/algorithms/dijkstras.pdf
	A* Search using inconsistent heuristic	AC T	Research Paper (Exact)	On the Complexity of Admissible Search Algorithms by Alberto Martelli
			Research Paper (Related)	http://web.cs.du.edu/~sturtevant/papers/incaaai.pdf
			Research Paper (Related)	https://webdocs.cs.ualberta.ca/~holte/Publications/ijcai09.pdf
Malware Analyzer	Divide by 0 results in arithmetic exception and disk utilization	AC S	CWE-369 (Related)	https://cwe.mitre.org/data/definitions/369.html
			CVE-2016-3623 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-3623
			CVE-2016-5241 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5241
AirPlan, WithMi	Improperly padded encrypted packets leak information about unencrypted length	SC S	CVE-2012-4930 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4930
			CVE-2010-4007 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4007
			CVE-2008-2780 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2780
			CWE-780 (Related)	https://cwe.mitre.org/data/definitions/780.html
Linear Algebra Platform	User input controls distribution of computation tasks between threads	AC T	Research Paper (Related)	https://pdfs.semanticscholar.org/1226/f6571f0c298ce98a44f788645df0b386ec9e.pdf
Tour Planner, BidPal, PowerBroker	Side effect vulnerability leads to information leakage	SC T	Research Paper (Somewhat Related)	http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=086C78EB7C42FDC05D9A2A0CB6D19E05?doi=10.1.1.672.544&rep=rep1&type=pdf

			Research Paper (Related)	https://eprint.iacr.org/2009/538.pdf
SmartMail and WithMi	File I/O leads to information leakage	SC T	Presentation (Related)	http://2013.zeronights.org/includes/docs/Ivan_Novikov_-_Filesystem_timing_attacks_practice.pdf
Tweeter	Spell correction, complex looping structure	AC T	Research Paper (Related)	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.7929&rep=rep1&type=pdf
			Research Paper (Related)	http://www.inrg.csie.ntu.edu.tw/algorithm2014/homework/Wagner-74.pdf
			Research Paper (Somewhat Related)	https://pdfs.semanticscholar.org/0517/aa6d420f66f74bd4b281e2ed0e2021f3d359.pdf
			Research Paper (Related)	https://pdfs.semanticscholar.org/4292/24a1c272168aa26bdf69e4de14863caa2e85.pdf

Note: The distinctions of *Exact*, *Related*, and *Somewhat Related* are used to highlight the relationship of the real-world examples to the referenced STAC vulnerabilities. *Exact* denotes cases where the referenced example is directly equivalent to the STAC vulnerability (ies); *Related* denotes cases where the referenced example is closely related to the STAC vulnerability (ies); *Somewhat Related* denotes cases where the reference example is loosely related to the STAC vulnerability(ies).

4.3.2.2.2 CyberPoint

CyberPoint produced 16 challenge programs with 38 questions attributed to the different challenge programs. The following table breaks down the number of intended positive and null questions for each of CyberPoint's challenge programs.

Table 17: Engagement 3 CyberPoint Programs and Questions

Challenge Program	# of Intended Positive Qs	# of Intended Null Qs	Total # of Qs	# of Attempts	# of Correct Attempts*	Percentage Correct
AirPlan_1	1	1	2	6	4	67%
AirPlan_2	2	0	2	5	3	60%
AirPlan_3	2	1	3	5	3	60%
AirPlan_4	2	1	3	4	2	50%
BidPal_1	1	1	2	4	2	50%
BidPal_2	2	0	2	3	0	0%
PowerBroker_1	2	0	2	2	1	50%
PowerBroker_2	1	1	2	2	1	50%
PowerBroker_3	1	2	3	2	1	50%
PowerBroker_4	1	0	1	1	1	100%
WithMi_1	2	1	3	7	5	71%
WithMi_2	2	0	2	2	2	100%
WithMi_3	1	1	2	5	5	100%
WithMi_4	4	0	4	9	7	78%
WithMi_5	1	2	3	4	2	50%
WithMi_6	1	1	2	2	2	100%
Total	25	13	38	63	41	65%

*Includes correct “yes” answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

Overall, the Blue Teams answered 63 CyberPoint questions with 41 of them being correct (this results in an overall accuracy of 65%). Blue teams achieved accuracy of 100% for PowerBroker 4, WithMi variants 2, 3 and 6 respectively. Blue teams struggled with BidPal variant 2 with an accuracy of 0%.

4.3.2.2.3 Raytheon/BBN Technologies

BBN produced 8 challenge programs with 19 questions attributed to the different challenge programs. The following table breaks down the questions asked between intended positive and null questions.

Table 18: Engagement 3 Raytheon/BBN Technologies Programs and Questions

Challenge Program	# of Intended Positive Qs	# of Intended Null Qs	Total # of Qs	# of Attempts	# of Correct Attempts*	Percentage Correct
Collab	0	3	3	12	8	67%
Info Trader	2	0	2	8	3	38%
Linear Algebra Platform	2	0	2	6	3	50%
Malware Analyzer	1	1	2	8	4	50%
RSA Commander	1	1	2	5	3	60%
SmartMail	1	2	3	6	3	50%
Tour Planner	1	2	3	4	1	25%
Tweeter	2	0	2	5	3	60%
Total	8	11	19	54	28	52%

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

In total, 54 BBN questions were answered (24 of which were correctly answered) with an accuracy of 52%. Of BBN's challenge programs, Collab had the most attempts and the highest accuracy, 67%. Blue teams struggled most with the Info Trader, Tour Planner, and Linear Algebra Platform challenge programs.

4.4 Engagement 4

4.4.1 Engagement Plan

4.4.1.1 Purpose and Program Scope

4.4.1.1.1 Program Scope

This Engagement Plan is for Engagement 4 which is a take-home engagement. All engagement answers must be submitted by noon Eastern time on June 28th, 2017. Answers should be sent to evan.fortunato@apogee-research.com or uploaded to Apogee's Gitlab server.

The document is structured into the following sections.

Section 1: Purpose and Program Scope
Section 2: Operational Definitions and Research Hypotheses
Section 3: Engagement 3 Plans and Logistics
Section 4: Post-Engagement Analysis Plans

4.4.1.2 Operational Definitions and Program Hypotheses

4.4.1.2.1 Operational Definitions of STAC Vulnerabilities

To guide the Engagements, the STAC AC, EL, and government teams have come up with

operational definitions of vulnerability to algorithmic complexity attack and to side channels. The AC performers will use these definitions to determine when the vulnerabilities they've inserted into their challenge programs are strong enough for use in engagements. The Control and R&D Teams will use these operational definitions to determine when they've found a vulnerability worth reporting – i.e., if it meets the definition of a STAC vulnerability. The operational definition document, currently *2016-06-1-opdef-v07.1*, specifies the definitions for STAC vulnerabilities along with guidance for Blue team responses to engagement challenges.

4.4.1.2.1.1 Unintended Vulnerabilities in Challenge Programs

Many Algorithmic Complexity and Side Channel vulnerabilities may exist in each challenge program beyond those intentionally inserted. For example, a normal authentication function is technically a side channel that leaks a secret. However, most of these vulnerabilities are "weak" – e.g., for Algorithmic Complexity, the additional complexity is a small percentage of the normal complexity, and for Side Channels, the number of operations necessary to resolve the secret is impractically large.

The defined operational definitions defined above are designed to minimize these unintended vulnerabilities in the challenge programs with the use of strong vulnerability thresholds (e.g., adversary operation budget is small for Side Channels). These thresholds will serve to discount most of these "unintended" vulnerabilities as not meeting the STAC definition. However, some unintended vulnerabilities will occur that do meet the definition. A reported actual vulnerability that satisfies the operational definition criterion for a given challenge program is a correct response, regardless if it is an intended or unintended vulnerability on behalf of the AC teams. During the analysis period after each Engagement, the EL team will work with the R&D and Control performers to verify that reported unintended vulnerabilities do actually meet the operational definition so that full credit is given. Unintended vulnerabilities can fall into two categories: those "in-scope" as per the STAC Operational Definitions document and those "out-of-scope" as per the STAC Operational Definitions document. Full credit will be given for all verified unintended vulnerability whether they are "in-scope" or "out-of-scope." In the case of "in-scope" unintended vulnerabilities, the engagement answer key will be updated to reflect the identified unintended vulnerabilities.

4.4.1.2.2 Research Hypothesis and Segmentation of Challenge Problems

Throughout the STAC program, the EL team will have a set of research hypotheses pertaining to performance of the tools with respect to different types of STAC vulnerabilities and challenge problems. Example hypotheses include:

10. Complexity vulnerabilities can be due to: (1) a change in the complexity of the algorithm, (2) a change in the coefficients of a fixed complexity, or (3) a flawed data guard that allows processing of input data that shouldn't be allowed. How do the different R&D techniques perform with respect to each of these types of complexity vulnerabilities?
11. Side channels can be weakened by increasing the number of secret symbols that are mapped to identical observed symbols or by increasing the width of the parent distributions of the observed symbols (causing overlaps of the of the samples from different observed symbol distributions). How do the different R&D techniques perform with respect to each of these challenges?
12. Some non-vulnerable programs are locally balanced (e.g., all paths through a sub-section of the binary consume equivalent time and space resources) while others are non-locally

balanced (e.g., resource consumption is asymmetric on a set of paths in multiple parts of the program resulting in entire execution traces that are balanced). How do the different R&D techniques perform with respect to the locality of balancing of resource consumption?

In support of these hypotheses, each challenge problem will be assessed and categorized with respect to how results on the challenge problem will support resolution of each of the hypotheses. This segmentation of the challenge problems will not be exposed during the engagements, but will be tracked for post-engagement analysis.

During the engagements, each challenge problem will come questions such as:

13) Does the challenge problem contain a time-based side channel?
14) Does the challenge problem contain a space-based side channel?
15) Does the challenge problem contain a time-based complexity vulnerability?
16) Does the challenge problem contain a space-based complexity vulnerability?

Note, that the questions will be more specific version of these classes of questions including additional context information and could include questions regarding combined time and space vulnerabilities.

4.4.1.3 Engagement 4 Plans and Logistics

4.4.1.3.1.1 Engagement 4 Schedule

The schedule below provides an overview of the milestones to happen during the four months of Engagement 4, including the engagement itself, the site visits for post Engagement 3 analysis and the post engagement evaluation.

- Engagement 4 (all STAC teams): Event runs from 3/3/2017- 6/28/2017.
- Release Engagement Plans (EL team): Initial release on 2/15/2017. Post on GitLab Public_EL_Information Project
- Obtain and Release Updated Challenge Programs (AC and EL Teams): The AC teams will provide directed updates to the challenge programs post Engagement 3 to the EL team by 2/15/2017 and the EL team will release these updates to the Blue Teams by 3/3/2017.
- Site Visits Post Engagement 3 (All STAC teams): 4/24/2017 through 5/12/2017 at each of the team sites.
- Quick-look Engagement 4 results (EL team): initial Engagement 4 quick-look results released at the July PI meeting. Results will subsequently be posted to each team's repo on GitLab: 8/2/2017.
- Teams will have until 8/30/2017 to request evaluation clarifications and changes.
- Final Engagement 3 analysis report delivered to DARPA and the AC teams: 9/28/2017

4.4.1.4 Post Engagement Analysis Plans

4.4.1.4.1.1 Metrics

We will be using the following set of metrics for Engagement 4:

- True Positive Rate (Number of correctly identified programs as vulnerable / Total number of vulnerable programs)

- True Negative Rate (Number of correctly identified programs as not vulnerable / Total number of non-vulnerable programs).

Note, True Positive Rate is equivalent to a Probability of Detection Metric while the True Negative Rate is equal to 1 – False Alarm Rate. Using the True Positive and Negative Rate gives insight into the relative strengths of each of the teams with respect to finding and ruling out vulnerabilities but doesn't give us insight into the speed of the teams.

To account for the speed, we will consider two different segmentations of the data. In the first segmentation, only properly justified answers will be considered (in the numerator) and only answered challenge programs will be considered (in the denominator). This will allow us to see the performance of each approach on the set of problems for which they chose to answer. In the second segmentation, properly justified answers will be considered (in the numerator) and all challenge programs will be considered (in the denominator). This second segmentation will show the benefits of speed as teams that only answer a small fraction of the questions will not be able to score well under the second segmentation.

To achieve the segmentation, each team's answers will be evaluated against the intended vulnerabilities of the challenge programs. In cases where a team believes that there is a vulnerability when none was intended, the EL team will re-evaluate the challenge application (using the justification provided by the team) to determine if an unintended vulnerability is present in the challenge application. If there is an unintended vulnerability, the EL will determine there is an in-scope unintended vulnerability. If there is an in-scope unintended vulnerability, then the challenge program status will be updated to act as though that unintended vulnerability was intended, and all teams' answers will be rescored against this updated status. If there is an unintended out-of-scope vulnerability, then the status of the challenge program will not be updated (so other teams' scores will not change) but we will score the team that found the unintended out-of-scope vulnerability as if the challenge program in question is vulnerable. Note, this means that the denominators of the metrics for each team will be different even under the second data segmentation.

Next, for each team we will evaluate the justifications against the justification guidance provided. If the justification is sufficient, then the answer will be considered a justified answer and will be included in the justified answer analysis. Independent of the validity of the justification, if a question is answered, then the challenge program will be included in the denominator under the first data segmentation.

4.4.1.4.1.2 Initial Assessments, Refinements Based on Blue Team Feedback and Site Visits

At the July PI meeting, the EL team will provide an initial assessment of the performance of each team against the program metrics using the intended vulnerabilities (i.e., those vulnerabilities that were intentionally added to the challenge programs). Following the release of individual assessments to each team on 8/2/2017, teams will have until 8/30/2017 to request evaluation clarifications and changes.

4.4.2 Engagement Results

4.4.2.1 Overview

In Engagement 4 (E4), 59 questions were asked about 25 different challenge programs. Of the 25 challenge programs, 17 were provided by CyberPoint and 8 were provided by Raytheon/BBN Technologies (BBN). The 59 questions can be broken into 10 different categories. 5 different

question types were asked (Algorithmic Complexity (AC) in Time, AC in Space, Side Channel (SC) in Time, SC in Space, and SC in Space and Time). Table 19 provides the breakdown of the 59 questions asked.

Table 19: How many questions were asked for a given question type and intended question result

Question Type	Intended Question Result	Number of Questions
Algorithmic Complexity in Time	Positive	9
Algorithmic Complexity in Space	Positive	9
Side Channel in Time	Positive	5
Side Channel in Space	Positive	4
Side Channel in Time and Space	Positive	1
Algorithmic Complexity in Time	Null	7
Algorithmic Complexity in Space	Null	7
Side Channel in Time	Null	8
Side Channel in Space	Null	7
Side Channel in Time and Space	Null	2

Eight R&D Teams and one control team, collectively known as the Blue Teams, participated in the engagement. In total, the Blue Teams attempted 268 questions (with 229 of those being attempted by the R&D Team, subset of the Blue Teams). The total combined accuracy of the Blue Teams was 70% (66% for just the R&D Teams alone). Table 20 provides a listing of the teams that participated in Engagement 4.

Table 20: The different teams that participated in Engagement 4

Blue Teams	Research Teams	University of Colorado Boulder (UC Boulder)
		Draper Labs
		GrammaTech
		Iowa State University (ISU)
		Northeastern University (NEU)
		University of Maryland (UMD)
		University of Utah
		Vanderbilt University
	Control Team	Invincea
Red Teams		CyberPoint
		BBN

A breakdown of the number of questions answered for each category and the resulting accuracy following a review by the EL is shown in Table 21.

Table 21: The distribution of Blue Team responses across different categories

	Total Responses	Correct Reviewed Responses	Accuracy
SC Space	63	51	81%
SC Time	58	38	67%
SC Space/Time	5	4	80%
AC Space	64	42	66%
AC Time	78	53	68%

Of the 268 questions attempted, 78 of them were AC Time questions with a total accuracy of 68% (62% for the R&D Teams). Research teams had the highest accuracy on SC questions with 107 responses and a 74% accuracy versus 122 responses for AC questions and a 67% accuracy.

E4 contained a distribution of questions that were intended vulnerable and intended non-vulnerable. During the engagement, Blue Teams discovered unintended vulnerabilities within the challenge programs. These unintended vulnerabilities were classified as "in-scope" if they satisfied the STAC operational definition requirements, and "out-of-scope" if they did not. "In-scope" vulnerabilities verified by the EL led to changes in the classification of vulnerable and non-vulnerable questions for all Blue Teams. "Out-of-scope" vulnerabilities led to changes in the classification of vulnerable and non-vulnerable questions only for the Blue Team that identified the vulnerability. The resulting plots of accuracy versus number of responses are shown below.

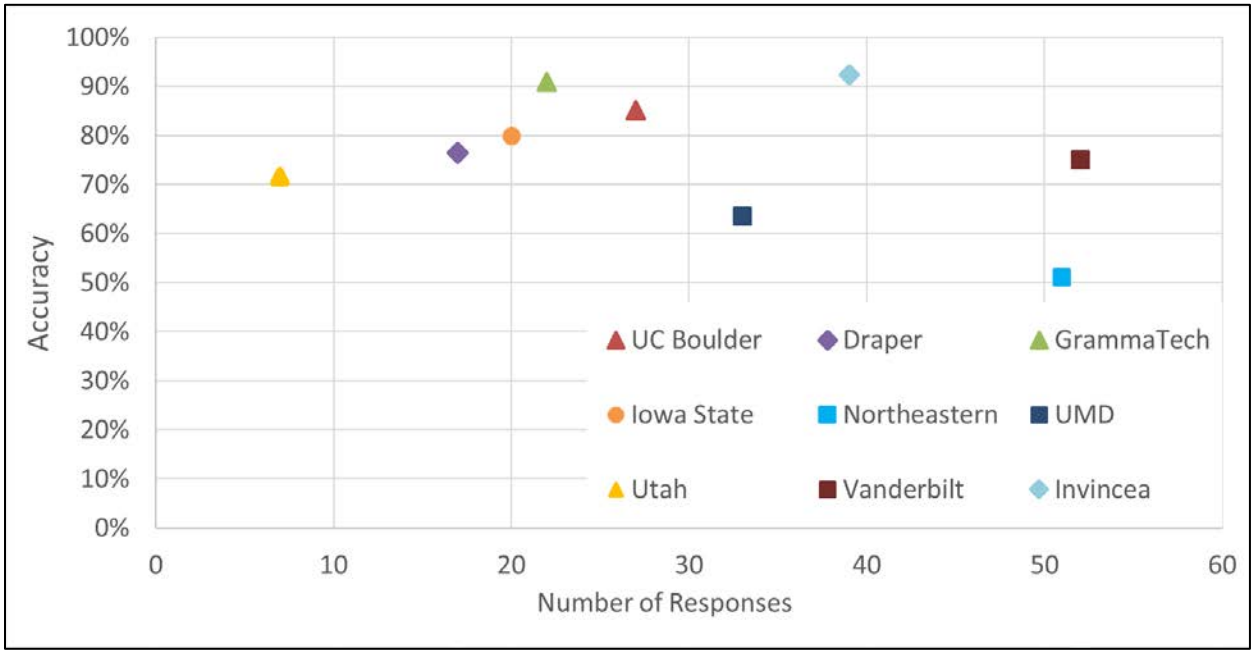


Figure 33: Raw accuracy versus number of responses for E4

The initial analysis of E4 responses used the raw responses provided by the Blue Teams. This analysis accounts for verified unintended vulnerabilities but not for justifications. Figure 33 provides the raw E4 results in the form of number answers provided versus overall accuracy.

Following this initial analysis, the EL reviewed the justifications provided by the Blue Teams to determine if sufficient justification for an identified vulnerability or lack of vulnerability was provided. Figure 34 provides the reviewed E4 results.

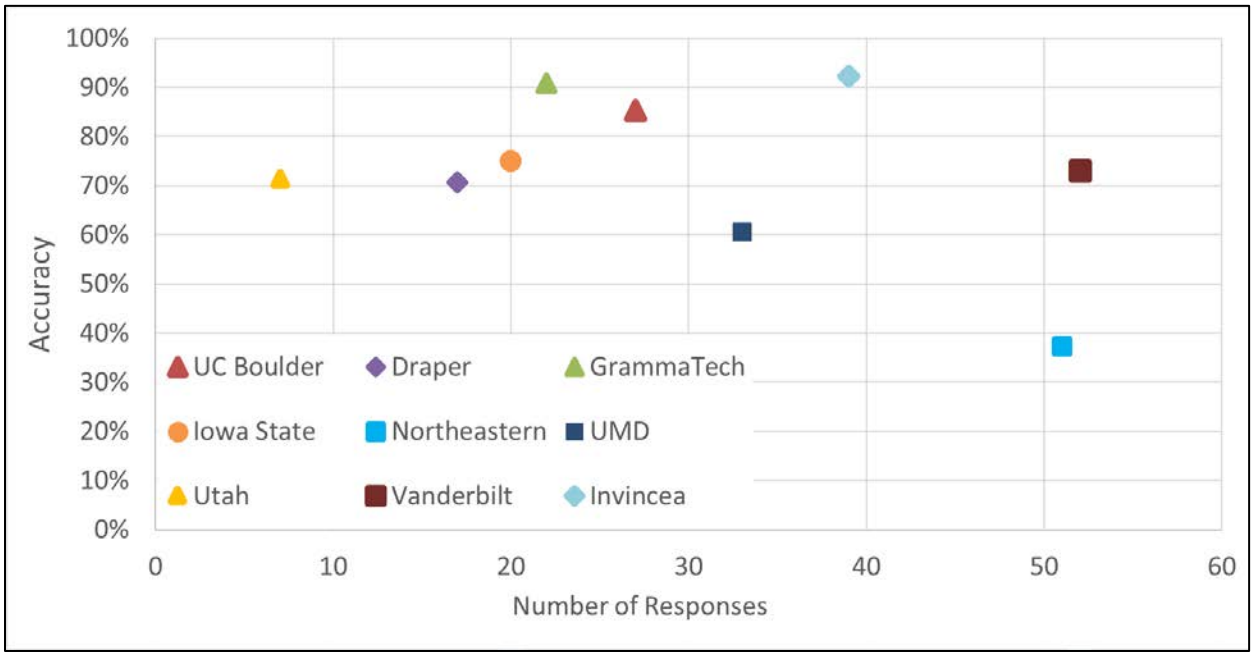


Figure 34: Reviewed accuracy versus number of responses for E4

The first group of Blue Teams, categorized by number of responses, is composed of Vanderbilt, Northeastern, and Invincea. Vanderbilt answered the most questions in E4, 52; Northeastern was a close second with 51 responses and Invincea was third with 39 responses. Accounting for the number of responses provided, Vanderbilt performed well with a 73% accuracy; however, Invincea achieved the highest overall accuracy of 92% for E4. UMD, Colorado and GrammaTech make up the second group of Blue Teams, with more than 20 responses, but fewer than 35. In this group, GrammaTech and Colorado have the highest accuracies, with GrammaTech having the highest accuracy across all R&D Teams of 91% and Colorado the second highest R&D Team accuracy of 85%. The third group of Blue Teams composed of Iowa State, Draper, and Utah answered 20 questions or fewer. In this group Iowa State leads with a 75% accuracy, with Draper and Utah tied at 71%.

Figure 35 shows the true positive rate (TPR – the ratio of the number of correct vulnerable responses to total number of vulnerable questions answered) versus the true negative rate (TNR – the ratio of the number of correct non-vulnerable responses to the total number of non-vulnerable questions answered). The data in Figure 35 is based on the reviewed responses for each Blue Team.

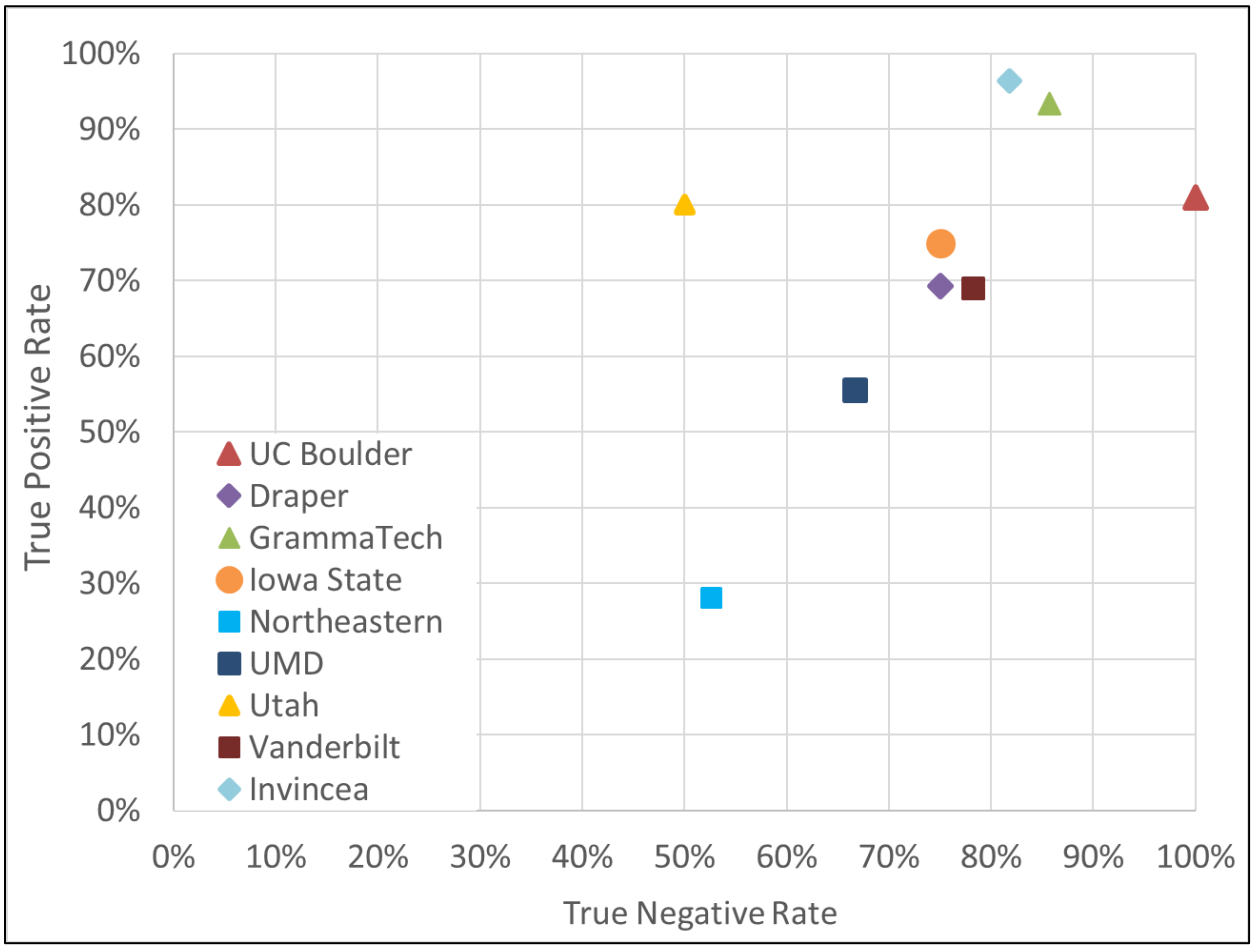


Figure 35: True positive rate (TPR) versus true negative rate (TNR) for the set of answered questions for E4

The TPR and TNR data for the set of answered questions separates the Blue Teams into three groups. The highest performing group in this segmentation is composed of Colorado, GrammaTech, and Invincea. Colorado has the highest overall TNR in this segmentation and Invincea the highest TPR. All three teams outperform all other Blue Teams in both TPR and TNR. The second group of Blue Teams is made up of Vanderbilt, Iowa State, and Draper. In this group Vanderbilt has the highest TNR and Iowa State the highest TPR. The third group of Blue Teams is composed of Utah, UMD, and Northeastern. This segmentation highlights each team's performance over their set of preferred questions; however, as it does not account for the number of responses, it does not capture the performance of each team over the full set of challenge questions presented at the engagement. To highlight this, Figure 36 shows the TPR versus TNR for the full set of questions for E4.

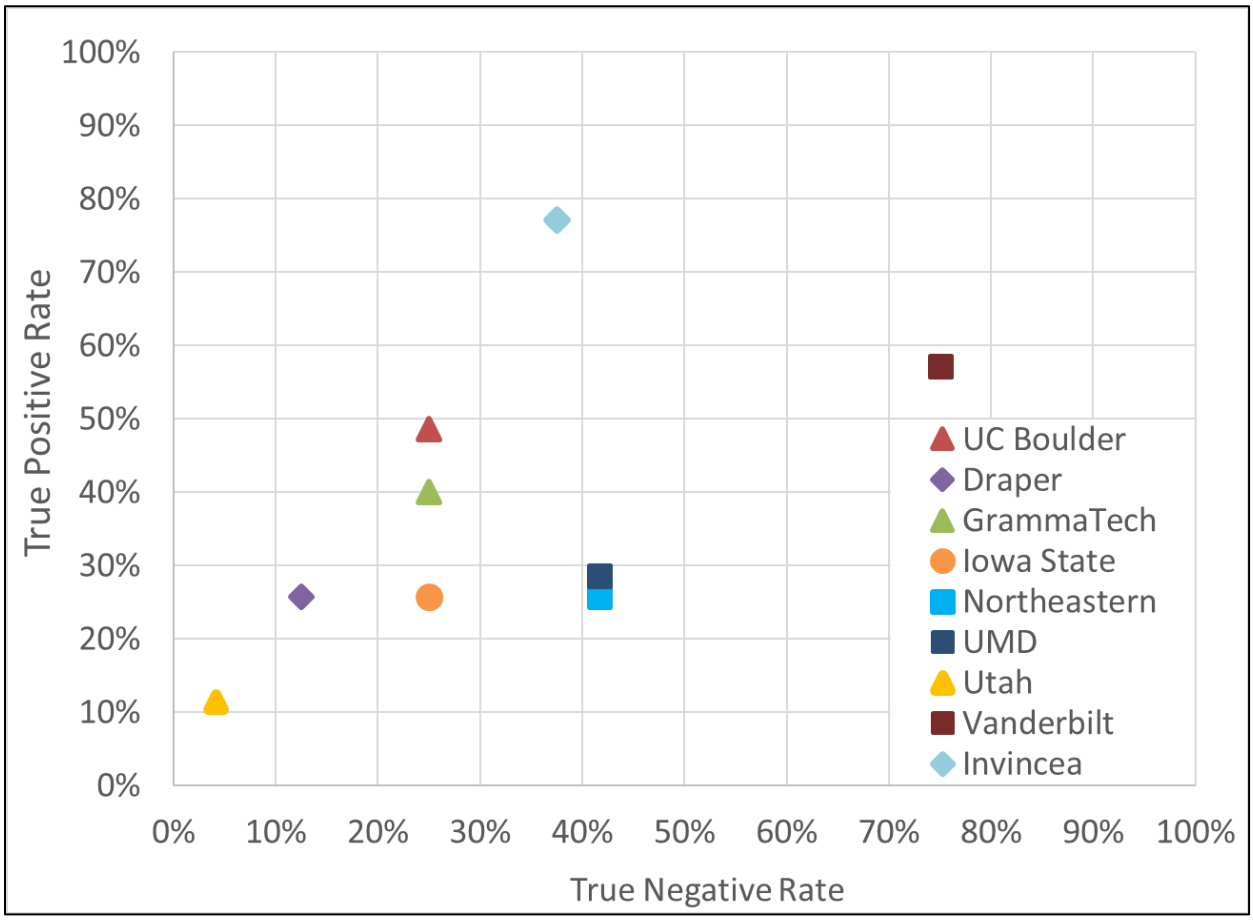


Figure 36: True positive rate (TPR) versus true negative rate (TNR) for the set of all questions for E4

Against the full set of challenge questions in E4 there are three groups of Blue Teams. The first group is composed of Invincea and Vanderbilt with Invincea having the highest TPR (77%) in this segmentation and Vanderbilt having the highest TNR (75%). Vanderbilt is the only team in this segmentation with both their TPR and TNR above 50%. The next group is composed in Colorado, GrammaTech, UMD, and Northeastern. Members of this group have either their TPR or TNR greater than 40%; note that in this segmentation no response is equivalent to an incorrect response. The third group is composed of Iowa State, Draper, and Utah. Members of this group answered the fewest number of questions in E4 and against the segmentation of all questions, their TPR and TNR fall below 30%.

The following tables show E4 responses for each team. Correct answers are in green and incorrect ones are in yellow.

Program	Question	Type	Vuln?	Northeastern	UC Boulder	GrammaTech	Draper
airplan_1	Question_020	AC Time	Y	-1	1	1	1
airplan_1	Question_051	AC Space	Y	-1	1	0	1
airplan_2	Question_037	AC Time	Y	-1	1	1	0
airplan_2	Question_038	SC Space	Y	1	1	1	0
airplan_3	Question_002	SC Space	Y	1	1	1	0
airplan_3	Question_016	SC Space	N	-1	1	1	0
airplan_3	Question_024	AC Time	Y	-1	1	0	1
airplan_4	Question_022	AC Time	Y	-1	1	1	1
airplan_4	Question_055	SC Space	N	1	1	1	0
airplan_5	Question_005	SC Space	N	1	1	1	0
airplan_5	Question_052	AC Space	Y	0	1	0	1
bidpal_1	Question_004	SC Time	N	-1	0	0	-1
bidpal_1	Question_033	AC Time	Y	-1	-1	0	1
bidpal_2	Question_044	AC Time	Y	-1	1	0	1
bidpal_2	Question_057	SC Time	Y	-1	-1	0	0
collab	Question_028	SC Time	N	-1	0	1	0
collab	Question_035	AC Time	N	1	0	0	1
collab	Question_049	SC Space	N	1	0	0	0
info_trader	Question_008	AC Time	N	-1	0	0	0
info_trader	Question_042	AC Space	Y	-1	0	0	0
linear_algebra_platform	Question_034	AC Space	Y	-1	0	1	0
linear_algebra_platform	Question_050	AC Time	Y	-1	1	0	0
malware_analyzer	Question_023	AC Space	Y	-1	-1	-1	-1
malware_analyzer	Question_058	AC Time	Y	-1	0	0	-1
powerbroker_1	Question_019	AC Space	Y	1	1	0	1
powerbroker_1	Question_040	SC Time	Y	-1	0	0	0
powerbroker_2	Question_015	SC Time	N	-1	0	0	0
powerbroker_2	Question_025	AC Space	Y	1	1	0	0
powerbroker_3	Question_001	SC Time/Space	N	0	0	0	0
powerbroker_3	Question_003	SC Time/Space	N	0	0	0	0
powerbroker_4	Question_006	SC Time	N	1	0	0	0
powerbroker_4	Question_013	SC Time	Y	0	0	0	0
rsa_commander	Question_039	AC Time	Y	-1	0	0	-1
rsa_commander	Question_053	SC Time	N	-1	0	0	1
smartmail	Question_043	AC Space	N	1	0	1	0
smartmail	Question_047	SC Time/Space	Y	0	0	0	1
smartmail	Question_059	AC Time	Y	1	0	0	-1
tour_planner	Question_011	SC Space	N	-1	1	1	0
tour_planner	Question_045	SC Time	Y	-1	-1	0	0
tour_planner	Question_056	AC Time	N	1	0	0	1
tweeter	Question_021	AC Space	Y	1	1	0	0
tweeter	Question_029	AC Time	Y	-1	0	1	0
withmi_1	Question_007	AC Space	N	1	0	0	0
withmi_1	Question_017	AC Time	Y	-1	0	1	0
withmi_1	Question_032	SC Space	Y	-1	0	1	0
withmi_2	Question_012	SC Space	N	0	1	0	0
withmi_2	Question_018	AC Space	Y	-1	0	0	0
withmi_3	Question_026	SC Time	N	1	0	0	0
withmi_3	Question_030	AC Space	Y	1	0	1	0
withmi_4	Question_010	SC Space	Y	1	1	1	0
withmi_4	Question_031	SC Time	Y	1	1	0	0
withmi_4	Question_036	AC Space	Y	-1	1	1	0
withmi_4	Question_041	SC Time	N	-1	0	0	0
withmi_5	Question_009	AC Space	N	0	0	0	0
withmi_5	Question_014	AC Space	Y	-1	1	1	0
withmi_5	Question_027	SC Time	N	1	1	0	0
withmi_5	Question_046	SC Space	N	-1	0	-1	0
withmi_6	Question_048	AC Space	N	0	0	0	0
withmi_6	Question_054	AC Time	Y	-1	0	1	0

Figure 37: Correct (green), incorrect (red), and unaswered (yellow) questions from engagement part 1

Program	Question	Type	Vuln?	Utah	Iowa State	UMD	Vanderbilt	Invincea
airplan_1	Question_020	AC Time	Y	0	0	0	1	1
airplan_1	Question_051	AC Space	Y	0	0	0	0	0
airplan_2	Question_037	AC Time	Y	1	1	0	1	1
airplan_2	Question_038	SC Space	Y	0	1	1	1	1
airplan_3	Question_002	SC Space	Y	0	1	1	1	-1
airplan_3	Question_016	SC Space	N	0	1	1	1	0
airplan_3	Question_024	AC Time	Y	0	0	0	1	1
airplan_4	Question_022	AC Time	Y	0	0	0	1	1
airplan_4	Question_055	SC Space	N	0	1	1	1	0
airplan_5	Question_005	SC Space	N	0	1	-1	1	0
airplan_5	Question_052	AC Space	Y	0	0	0	0	1
bidpal_1	Question_004	SC Time	N	0	0	0	1	0
bidpal_1	Question_033	AC Time	Y	0	0	0	1	1
bidpal_2	Question_044	AC Time	Y	0	0	0	1	1
bidpal_2	Question_057	SC Time	Y	0	0	0	1	1
collab	Question_028	SC Time	N	-1	0	-1	-1	-1
collab	Question_035	AC Time	N	0	0	1	1	0
collab	Question_049	SC Space	N	1	0	0	1	1
info_trader	Question_008	AC Time	N	0	0	-1	1	0
info_trader	Question_042	AC Space	Y	0	0	0	-1	0
linear_algebra_platform	Question_034	AC Space	Y	1	0	1	0	1
linear_algebra_platform	Question_050	AC Time	Y	0	0	1	1	1
malware_analyzer	Question_023	AC Space	Y	-1	-1	1	1	0
malware_analyzer	Question_058	AC Time	Y	0	1	-1	-1	1
powerbroker_1	Question_019	AC Space	Y	0	1	1	-1	1
powerbroker_1	Question_040	SC Time	Y	0	0	1	1	1
powerbroker_2	Question_015	SC Time	N	0	-1	1	1	1
powerbroker_2	Question_025	AC Space	Y	0	1	-1	-1	1
powerbroker_3	Question_001	SC Time/Space	N	0	0	0	1	0
powerbroker_3	Question_003	SC Time/Space	N	0	0	0	1	0
powerbroker_4	Question_006	SC Time	N	0	0	1	1	1
powerbroker_4	Question_013	SC Time	Y	0	0	-1	1	1
rsa_commander	Question_039	AC Time	Y	0	0	0	-1	1
rsa_commander	Question_053	SC Time	N	0	0	0	1	-1
smartmail	Question_043	AC Space	N	0	0	0	1	0
smartmail	Question_047	SC Time/Space	Y	0	1	-1	0	0
smartmail	Question_059	AC Time	Y	1	-1	0	-1	0
tour_planner	Question_011	SC Space	N	0	0	0	-1	1
tour_planner	Question_045	SC Time	Y	0	0	-1	1	1
tour_planner	Question_056	AC Time	N	0	1	0	1	0
tweeter	Question_021	AC Space	Y	0	-1	0	1	1
tweeter	Question_029	AC Time	Y	1	0	0	1	1
withmi_1	Question_007	AC Space	N	0	0	1	-1	0
withmi_1	Question_017	AC Time	Y	0	0	0	-1	1
withmi_1	Question_032	SC Space	Y	0	1	1	1	0
withmi_2	Question_012	SC Space	N	0	1	-1	1	1
withmi_2	Question_018	AC Space	Y	0	0	0	-1	0
withmi_3	Question_026	SC Time	N	0	0	1	1	1
withmi_3	Question_030	AC Space	Y	0	0	1	1	1
withmi_4	Question_010	SC Space	Y	0	0	-1	1	1
withmi_4	Question_031	SC Time	Y	0	1	1	1	1
withmi_4	Question_036	AC Space	Y	0	0	-1	0	1
withmi_4	Question_041	SC Time	N	0	0	-1	1	1
withmi_5	Question_009	AC Space	N	0	0	1	0	0
withmi_5	Question_014	AC Space	Y	0	0	-1	0	1
withmi_5	Question_027	SC Time	N	0	1	1	1	1
withmi_5	Question_046	SC Space	N	0	-1	1	-1	1
withmi_6	Question_048	AC Space	N	0	0	0	-1	0
withmi_6	Question_054	AC Time	Y	0	0	0	-1	1

Figure 38: Correct (green), incorrect (red), and unaswered (yellow) questions from engagement part 2

4.4.2.2 Overall Trends

To highlight the overall trend in accuracy between engagements, the plots below show the data for live engagements (E1 and E3), and take-home engagements (E2 and E4). The arrows in these plots indicate the direction of the lines from older to newer engagements.

4.4.2.2.1 Accuracy and Number of Responses

Figure 39 and Figure 40 show the trends in accuracy versus number of responses for live and take-home engagements respectively.

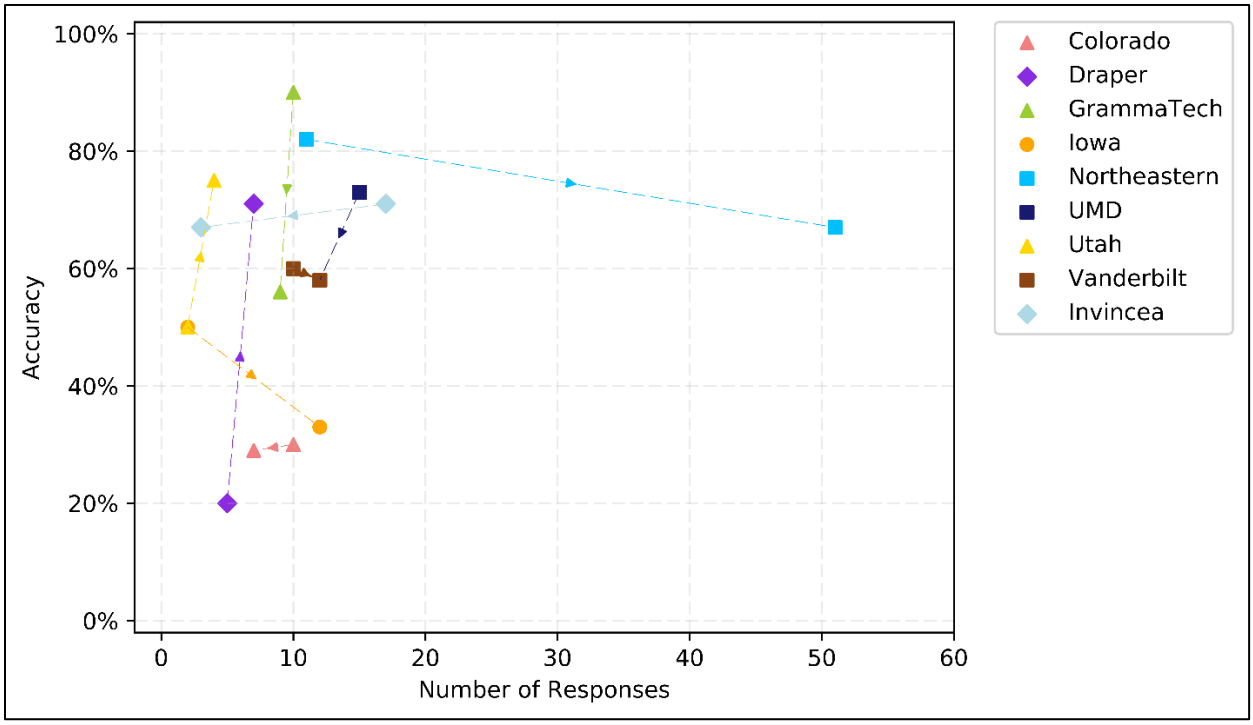


Figure 39: Plot of live engagement trends in accuracy versus number of responses

The data for live engagements from E1 to E3 shows that all Blue Teams except for Colorado, GrammaTech, UMD, and Invincea provided more responses for E3 than for E1. Utah and Draper are the only Blue Teams with a higher total accuracy for E3 than for E1. Northeastern had the greatest increase in the number of responses between the two engagements and Draper the greatest increase in accuracy. Several unintended vulnerabilities that impacted 7 challenge questions were identified by multiple teams in E1, in contrast during E3 Northeastern identified multiple unintended vulnerabilities that impacted 15 challenge questions in E3 accounting for 44% of their correct responses. GrammaTech had the greatest decrease in accuracy between the two engagements, and Invincea the greatest decrease in number of responses.

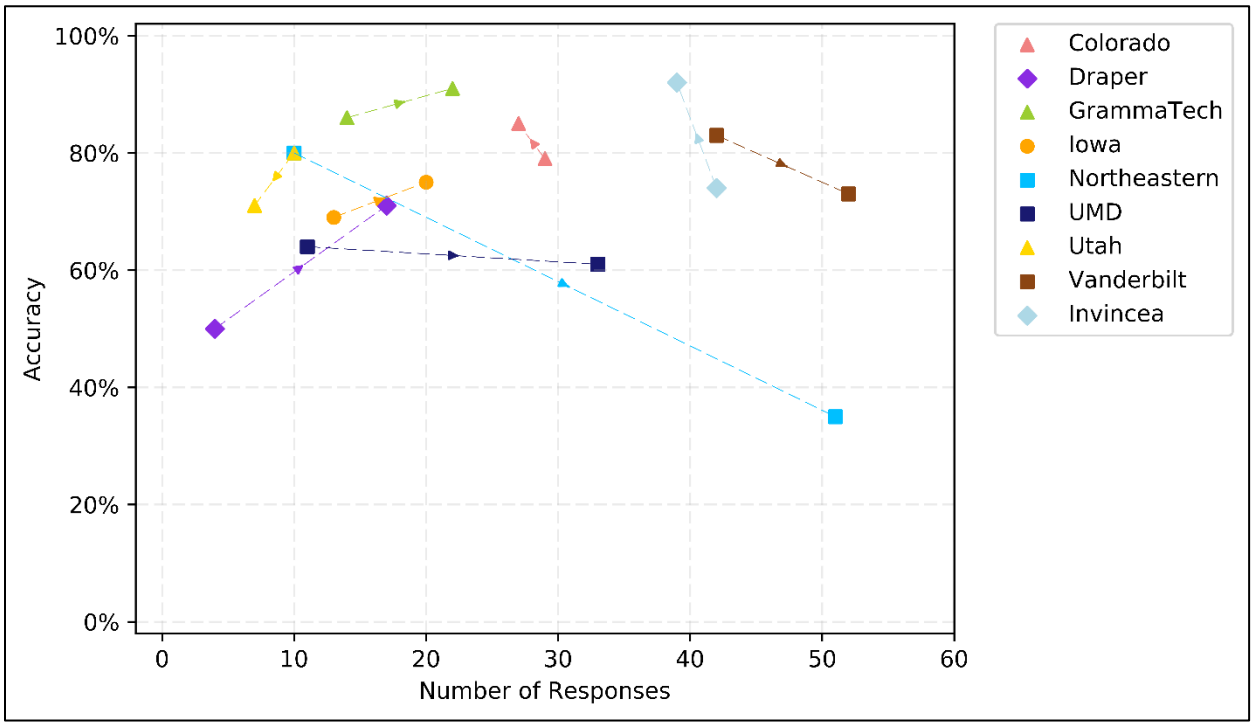


Figure 40: Plot of take-home engagement trends in accuracy versus number of responses

The data for take-home engagements shows an increase in the number of responses from E2 to E4 for all teams except for Colorado, Utah, and Invincea. GrammaTech had the highest accuracy for E2 and Invincea for E4. GrammaTech, Iowa, and Draper are the only teams to increase both their number of responses and accuracy from E2 to E4. Vanderbilt answered the greatest number of questions in both take-home engagements, tying with Invincea in E2 but with greater accuracy. Vanderbilt has performed better at take-home that live engagements in both accuracy and number of responses. Northeastern had the greatest increase in number of responses, but also the greatest decrease in accuracy between the two take-home engagements. Northeastern identified many unintended vulnerabilities in E3 that were removed for E4, Northeastern's performance difference in accuracy between E3 and E4 may be in part due to this.

4.4.2.2.2 Answered Questions TPR and TNR

Figure 41 and Figure 42 show the trends in TPR versus TNR against the segmentation of answered questions for live and take-home engagements respectively. Between the two live engagements against the segmentation of answered questions, no team increased in both TPR and TNR. All teams except for Invincea and Draper decreased in their ability to identify vulnerabilities (TPR) and 5 teams increase in their ability to rule out vulnerabilities (TNR), with 4 teams increasing from 0% TNR in E1. Draper is the only team to decrease in TNR between the two engagements, while Colorado had a 100-percentage point increase in TNR between the two engagements. Vanderbilt and Iowa have approximately the same TNR across live engagements, but with Vanderbilt maintaining a 25-percentage point advantage in TPR. Utah, GrammaTech, and Invincea are the only teams with a 0% TNR across all live engagements.

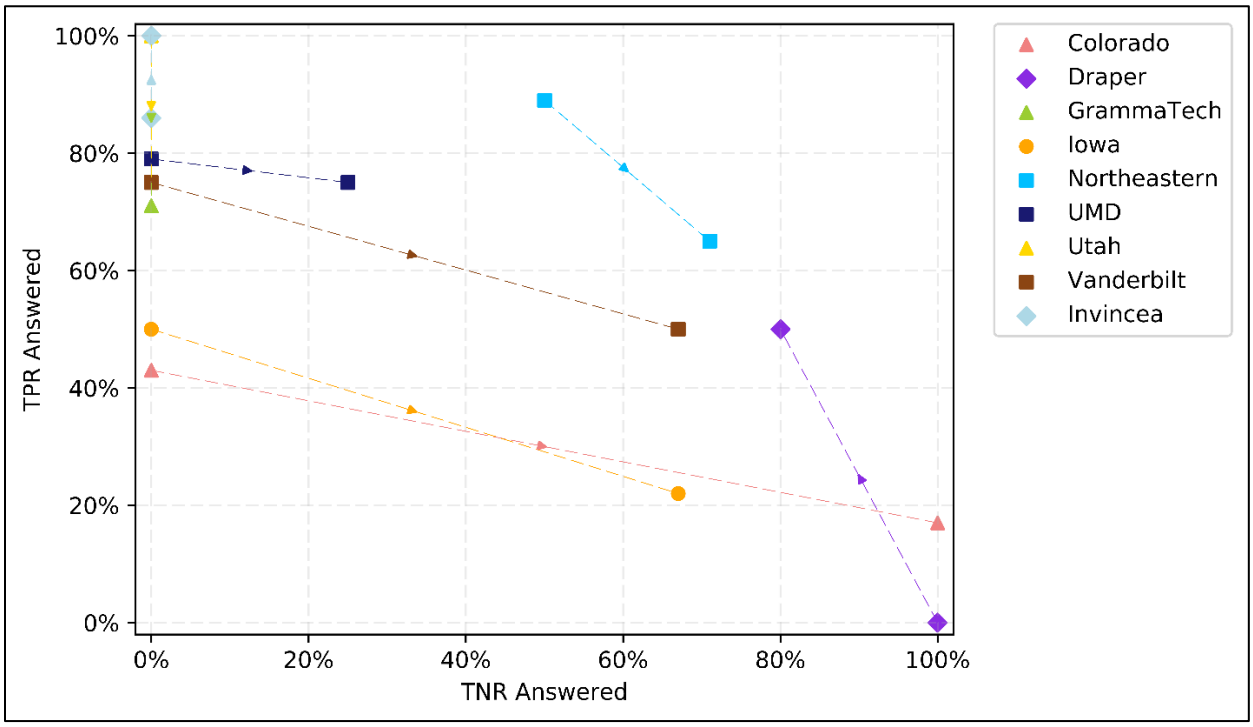


Figure 41: Plot of live engagement trends in TPR versus TNR against the segmentation of answered questions

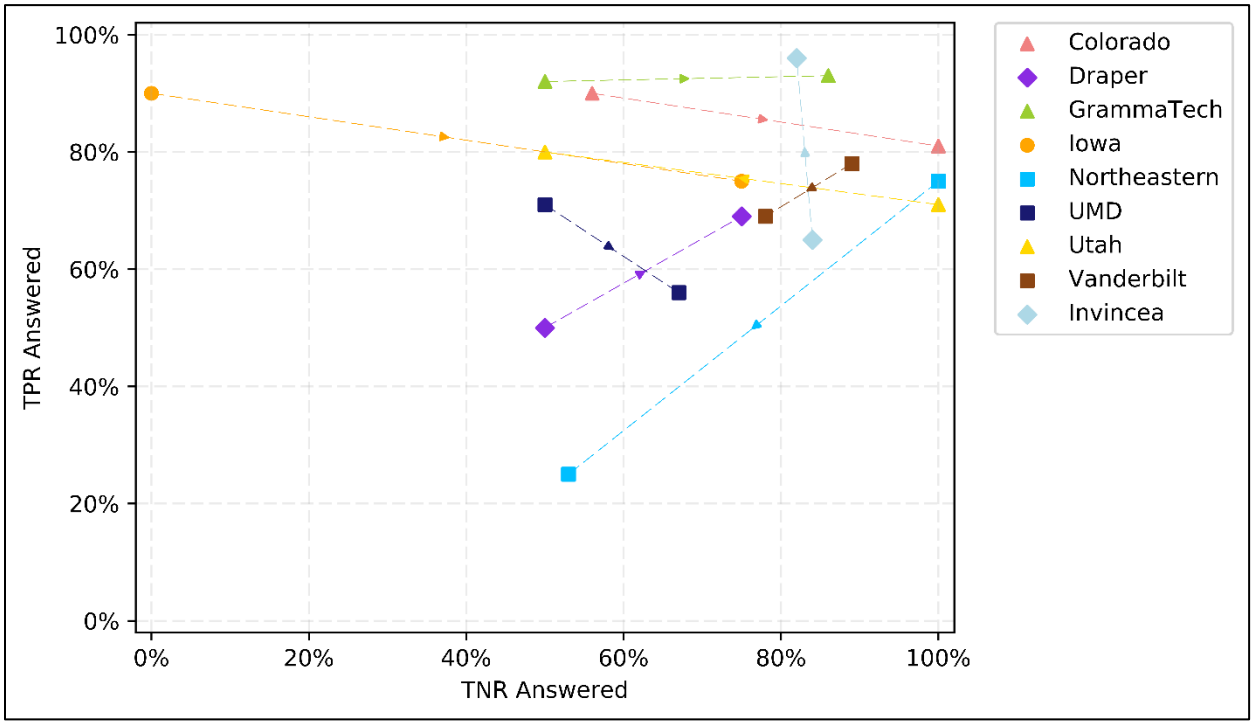


Figure 42: Plot of take-home engagement trends in TPR versus TNR against the segmentation of answered questions

TPR versus TNR data against the segmentation of answered questions for take-home engagements (Figure 42) show all teams but Iowa (E2) and Utah (E4) with both their TPR and TNR greater than or equal to 50%. Invincea was outperformed in TPR by all R&D Teams except for Draper in E2 and outperformed all R&D Teams except for GrammaTech in TPR in E4. GrammaTech had the highest TPR for E2 engagements, and had a 35-percentage-point increase in TNR between the two take-home engagements. Invincea had the highest TPR for E2. Vanderbilt and Northeastern were the only two teams to decrease in both TPR and TNR; however, Vanderbilt TPR and TNR decreased by 9 percentage points and 12 percentage points respectively while Northeastern's TPR and TNR both decreased by 47 percentage points. This segmentation does not account for number of responses, and Northeastern answered 41 more questions in E4 than in E2. Draper is the only teams to have increased in both TPR and TNR between the two engagements, but as is noted later, for several questions it is unclear how their research approach of identifying the upper and lower complexity bounds for individual methods and their tool provided an answer without a significant manual effort on the question.

4.4.2.2.3 All Questions TPR and TNR

Figure 43 and Figure 44 show the trends in TPR versus TNR against the segmentation of all questions for live and take-home engagements respectively.

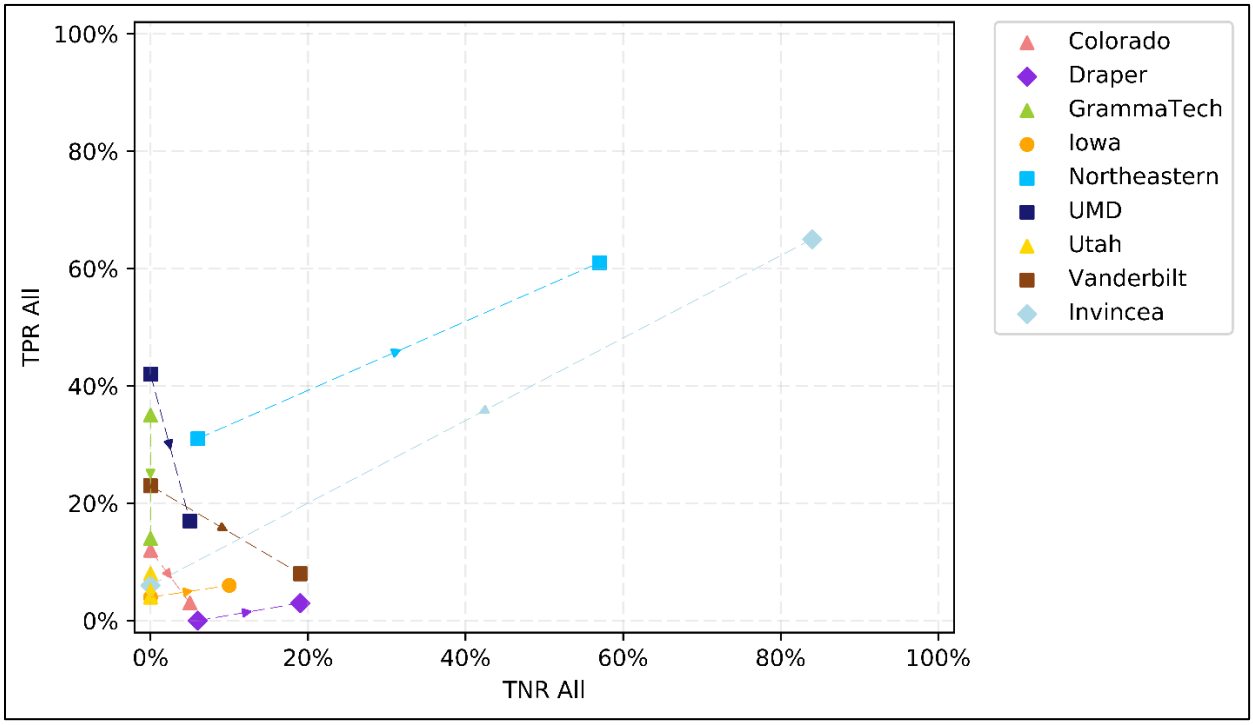


Figure 43: Plot of live trends in TPR versus TNR against the segmentation of all questions

In the segmentation of all questions for live engagements, Invincea's performance in E1, and Northeastern's performance in E2 stand out. In engagement 1, Invincea identified many unintended vulnerabilities, and in engagement 2 Northeastern did the same. Invincea had the greater decrease in performance between the two live engagements while Northeastern had the greatest increase in performance. In the segmentation of all questions, no other teams achieve a TNR above 20% or a TPR above 50%.

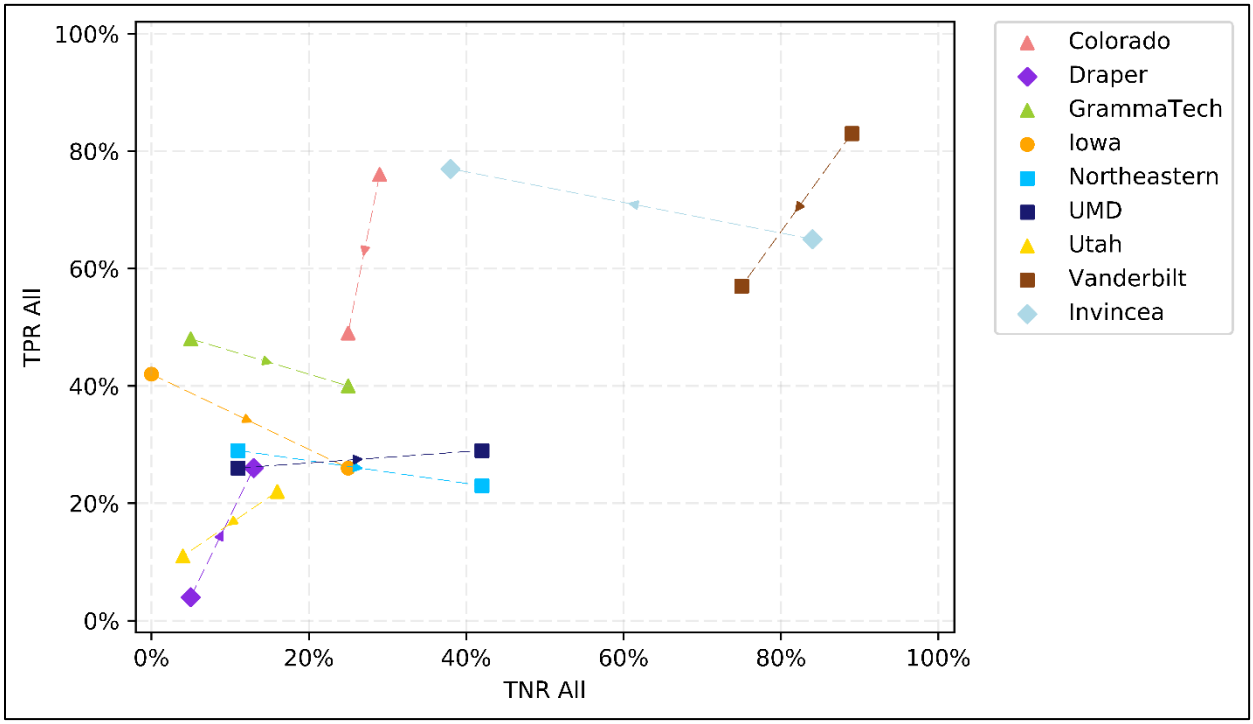


Figure 44: Plot of take-home engagement trends in TPR versus TNR against the segmentation of all questions

In the take-home engagements, Vanderbilt's E2 performance is unmatched by any Blue Team, and their TNR in both take-home engagements outperforms all other Blue Teams. Accounting for both TPR and TNR in this data segmentation, Colorado, Invincea, and Vanderbilt lead all other Blue Teams while Draper, and Utah are outperformed by all other Blue Teams.

4.4.2.2.4 Categories of Incorrect Responses

The incorrect responses by the Blue Teams for both AC and SC questions fall into four categories. These categories are not identical in both AC and SC questions, but they mirror each other. For AC questions, the incorrect Blue Team response categories are:

AC 1. Did not follow directions or violated the question definitions
AC 2. Failed to identify the complexity within the challenge program
AC 3. Failed to understand the input to the complexity control flow of the challenge program
AC 4. Failed to properly determine the complexity bounds of an identified complexity

For SC questions, the incorrect Blue Team response categories are:

SC 1. Did not follow directions or violated the question definitions
SC 2. Failed to identify the secret or the secret location within the challenge program
SC 3. Failed to identify the side channel (processing conditioned on the secret value)
SC 4. Failed to properly analyze the side channel strength

An incorrect response can fit into multiple categories e.g. given strong side channel a team identifies a weaker side channel. This is classified as both a failure to identify the intended side channel vulnerability and a failure to determine strength.

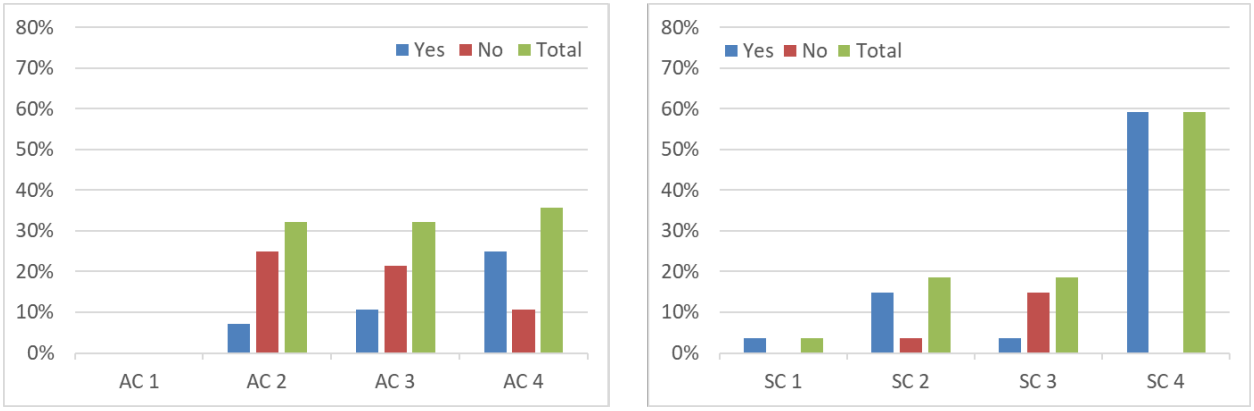


Figure 45: Plots of categories of incorrect responses in E3: algorithmic complexity questions (left), side channel questions (right)

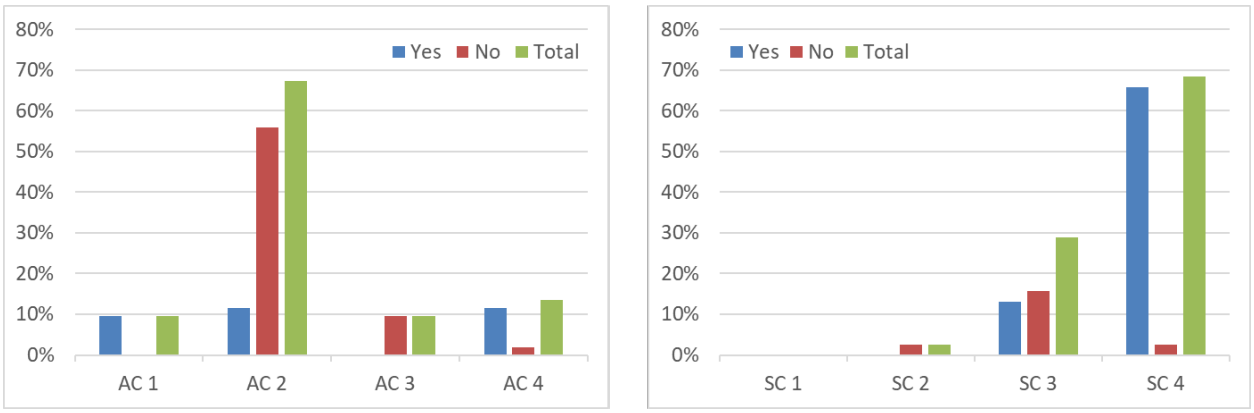


Figure 46: Plots of categories of incorrect responses in E4: algorithmic complexity questions (left), side channel questions (right)

The data shows that for AC questions, in E4, teams struggled to identify the complexity within the challenge applications. This resulted in teams incorrectly declaring the application non-vulnerable. This result may be supported by the fact that the removed unintended vulnerabilities from E3 left only more difficult AC vulnerabilities for E4. Failure to follow directions or read the challenge question accounted for 10% of the incorrect AC responses. These instances increased from 0% in E3. Failure to identify the input to the complexity control flow accounted for 10% of incorrect responses with teams uniformly incorrectly declaring applications non-vulnerable. This is in line

with the intuition that without identifying the input to the complexity control flow, teams are more likely to respond that a challenge program is non-vulnerable. Failure to analyze strength accounted for 14% of incorrect AC responses with teams more likely to incorrectly respond that a challenge program is vulnerable without properly assessing strength. In E4 for AC questions, the reasons for incorrect responses were similar to E3 in terms of likelihood of responding "Yes" or "No", but very different in that the reasons were heavily concentrated on an inability to identify the complexity rather than evenly distributed across AC 2, AC 3, and AC 4.

The data for SC questions shows that in E4 teams continued to have the most difficulty with evaluating strength, and were more likely to incorrectly declare an application vulnerable as a result. From E3 to E4 there was a significant decrease in the number of incorrect responses due to not identifying the secret location (SC 2) and an increase in incorrect responses due to not identifying the side channel (SC 3). In cases where teams failed to identify the side channel in E4, they were as likely to declare the application vulnerable as to declare it non-vulnerable.

4.4.2.3 AC Teams

4.4.2.3.1 Intended Vulnerabilities

The vulnerabilities in the STAC E4 challenge programs in addition to spanning space and time side channel and algorithmic complexity vulnerabilities contained components which can be mapped to real-world vulnerabilities. The table below presents general categorization of the E4 vulnerabilities and links to research papers, presentations, Common Vulnerabilities and Exposures (CVEs) and Common Weakness Enumerations (CWEs) that are comparable to the specified category.

Table 22: CVEs and CWEs mappings Engagement 4 challenges

Challenge Program (s)	Description	Vuln	Link Type	Links
Collab	Timing attacks on database indexing algorithms	SC T	Research Paper (Direct)	https://www.usenix.org/legacy/event/woot07/tech/full_papers/futoransky/futoransky.pdf
			Research Paper (Direct)	https://pdfs.semanticscholar.org/f100/b668ff5de9909919da47a85ac6f5ee019f22.pdf
PowerBroker, BidPal, WithMi	Timing attacks against RSA implementations using Montgomery multiplication	SC T	CVE-2013-5915 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2013-5915
			CVE-2004-2682 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2004-2682
			CVE-2003-0147 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2003-0147

			Research Paper (Direct)	https://tls.mbed.org/public/WSchindler-RSA_Timing_Attack.pdf
			Research Paper (Direct)	https://www.uclouvain.be/crypto/services/download/publications.pdf.9256d17be6b4cdd4.70646633362e706466.pdf
Info Trader	Tree traversal algorithm with broken guard to prevent cycle detection	AC S	Textbook (Related)	http://www.springer.com/cda/content/document/cda_downloaddocument/9783642381225-c2.pdf?SGWID=0-0-45-1402898-p175152334
Info Trader, Linear Algebra Platform, Airplan, RSA Commander, WithMi	Improper input validation resulting in a broken input guard	AC S/T	CWE-20 (Related)	https://cwe.mitre.org/data/definitions/20.html
			CVE-2017-3807 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-3807
			CVE-2010-3717 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-3717
Airplan, WithMi	Complexity vulnerabilities in compression implementations	AC S/T	CVE-2016-4630 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-4630
			CVE-2014-2746 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-2746
			CVE-2012-4447 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4447
Malware Analyzer, WithMi	Integer overflow vulnerability resulting in memory utilization	AC S	CVE-2016-1968 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1968
			CVE-2017-6308 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-6308
			CVE-2016-6207 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-6207
			CVE-2017-5627 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2017-5627

			CVE-2016-1933 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-1933
Linear Algebra Platform	Improper guard on input results in large memory allocation	AC S	CVE-2014-9192 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2014-9192
Airplan	Ford-Fulkerson vulnerable implementation	AC T	Textbook (Exact)	http://www.cse.unt.edu/~tarau/teaching/AnAlgo/Ford%E2%80%93Fulkerson%20algorithm.pdf
	Dijkstra's Algorithm vulnerable with negative weights	AC T	Textbook (Exact)	http://www.eecs.tufts.edu/~ccousi01/algorithms/dijkstras.pdf
	A* Search using inconsistent heuristic	AC T	Research Paper (Exact)	On the Complexity of Admissible Search Algorithms by Alberto Martelli
			Research Paper (Related)	http://web.cs.du.edu/~sturtevant/papers/incaaai.pdf
			Research Paper (Related)	https://webdocs.cs.ualberta.ca/~holte/Publications/ijcai09.pdf
Malware Analyzer	Divide by 0 results in arithmetic exception and disk utilization	AC S	CWE-369 (Related)	https://cwe.mitre.org/data/definitions/369.html
			CVE-2016-3623 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-3623
			CVE-2016-5241 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2016-5241
Airplan, WithMi	Improperly padded encrypted packets leak information about unencrypted length	SC S	CVE-2012-4930 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2012-4930
			CVE-2010-4007 (Somewhat Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2010-4007
			CVE-2008-2780 (Related)	https://cve.mitre.org/cgi-bin/cvename.cgi?name=CVE-2008-2780

			CWE-780 (Related)	https://cwe.mitre.org/data/definitions/780.html
Linear Algebra Platform	User input controls distribution of computation tasks between threads	AC T	Research Paper (Related)	https://pdfs.semanticscholar.org/1226/f6571f0c298ce98a44f788645df0b386ec9e.pdf
Tour Planner, BidPal, PowerBroker	Side effect vulnerability leads to information leakage	SC T	Research Paper (Somewhat Related)	http://citeseerx.ist.psu.edu/viewdoc/download;jsessionid=086C78EB7C42FDC05D9A2A0CB6D19E05?doi=10.1.1.672.544&rep=rep1&type=pdf
			Research Paper (Related)	https://eprint.iacr.org/2009/538.pdf
SmartMail and WithMi	File I/O leads to information leakage	SC T	Presentation (Related)	http://2013.zeronights.org/includes/docs/Ivan_Novikov_-_Filesystem_timing_attacks_practice.pdf
Tweeter	Spell correction, complex looping structure	AC T	Research Paper (Related)	http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.34.7929&rep=rep1&type=pdf
			Research Paper (Related)	http://www.inrg.csie.ntu.edu.tw/algorithm2014/homework/Wagner-74.pdf
			Research Paper (Somewhat Related)	https://pdfs.semanticscholar.org/0517/aa6d420f66f74bd4b281e2ed0e2021f3d359.pdf
			Research Paper (Related)	https://pdfs.semanticscholar.org/4292/24a1c272168aa26bdf69e4de14863caa2e85.pdf

Note: The distinctions of *Exact*, *Related*, and *Somewhat Related* are used to highlight the relationship of the real-world examples to the referenced STAC vulnerabilities. *Exact* denotes cases where the referenced example is directly equivalent to the STAC vulnerability(ies); *Related* denotes cases where the referenced example is closely related to the STAC vulnerability(ies); *Somewhat Related* denotes cases where the reference example is loosely related to the STAC vulnerability(ies).

4.4.2.3.2 Unintended Vulnerabilities

Collectively, Blue Teams identified 49 unintended vulnerabilities (41 unique – same vulnerability reported for same challenge question). All Blue Teams identified at least one

unintended vulnerability with Invincea identifying the greatest number, 13. 17 challenge questions in total were impacted by the unintended vulnerabilities. The 7 unintended vulnerabilities (6 unique) in BBN challenges impacted 4 questions (2 intended non-vulnerable). The 42 unintended vulnerabilities (35 unique) in CyberPoint challenges impacted 13 questions (5 intended non-vulnerable). The unintended vulnerabilities identified in Engagement 4 impacted 7 questions that were intended to be non-vulnerable. In some instances, multiple teams identified multiple vulnerabilities. The following provides a single instance of an unintended vulnerability for each of the impacted challenge questions intended to be non-vulnerable.

4.4.2.3.2.1 Airplan 1 Question 051 AC Space

Colorado and Draper both identified unintended vulnerabilities. Draper identified an unintended vulnerability in the challenge program. The application does not check for negative weights causing the optimal path algorithm to consume resources.

4.4.2.3.2.2 BidPal 1 Question 033 AC Time

Vanderbilt, Draper and Invincea identified unintended vulnerabilities. Invincea observed that "the bidpal_1 application does not verify a user's BidCommitmentData before using it to create ExchangeData. In the ExchangeData class, make() enters a while loop that is only terminated when verifySpacing(this.array, this.p) returns true. One of the first checks in verifySpacing() is to return false if any of the entries in the array equal 0. In the array passed to verifySpacing(), this.array[0] is equal to the decryption of commit.takeSharedValue() mod p. Note that if commit.takeSharedValue() is 0, then this.array[0] = 0 regardless of the value of p chosen. In this scenario, verifySpacing will always return false, and make() will be stuck in an infinite loop. By bidding on the same auction as other users, but sending BidCommitmentData with a sharedValue of 0 instead of an honest value, an attacker can cause any users who have bid on the auction, or who later bid on the auction, to be stuck in an infinite loop. In particular, having issued a bid commitment for an auction, an attacker will receive a bid receipt from all connected users, but will not receive the expected ExchangeData from users who have bid on the same auction within 1200 seconds of sending the BidCommitmentData. In fact, the ExchangeData will never arrive because the bidpal_1 application for affected users will be stuck in an infinite loop and unable to perform any further operations without being restarted."

4.4.2.3.2.3 Malware Analyzer Question 058 AC Time

Invincea identified an unintended vulnerability "due to a lack of input sanitization in the handling of a *PUT* request in the com.ainfosec.MalwareAnalyzer.Server.serve function within malware_analyzer.jar. When the com.ainfosec.MalwareAnalyzer.Server class receives a PUT HTTP request, it allocates a buffer that is defined by the size of the user-specified *content-length* field - this is done on line 496 of the com.ainfosec.MalwareAnalyzer.Server file. It then reads data from the request into this buffer and creates an MD5 sum of that buffer. Specifying a large *content-length* field, will cause a large delay, as the program is constantly updating an MD5 hash of a buffer that is very large."

4.4.2.3.2.4 PowerBroker 2 Question 025 AC Space

Colorado, Iowa State, Northeastern, and Invincea identified unintended vulnerabilities. Invincea's unintended vulnerability is presented. They observed that "the data structure used to hold the total value of generated power capacity […] does not check to see if the total will grow out of bounds before adding the next generation amount. This creates an unbounded situation

and makes an unlimited amount of sell actions. The list of unbounded sell actions is written to the output file […] which makes it grow to greater than" the resource usage limit.

4.4.2.3.2.5 SmartMail Question 059 AC Time

Northeastern and Utah identified the same unintended vulnerability. Northeastern observed that requests are sequentially processed and by sending multiple SmartMail messages with duplicate destinations of the longest mailing list the runtime for the final request exceeds the resource usage limit.

4.4.2.3.2.6 WithMi 5 Question 014 AC Space

Invincea identified an unintended vulnerability, noting that "the SpiffyWrapper::inflate() method doesn't do bound checking for if the input is a negative integer. […] one can have a compressed file:

| 00 | 60000000 |

| 00 | -60000000 |

| 00 | 60000000 |

inflate() would write 60000000 bytes, then it would adjust totalSize back to 0, then it would write 60000000 more bytes."

4.4.2.3.2.7 WithMi 6 Question 054 AC Time

GrammaTech and Invincea identified unintended vulnerabilities. Invincea identified an unintended vulnerability in the trie data structure used for decompression. They noted that "Each leaf in the trie is a data value, and during decompression a bitstream is used to traverse the trie. Trie traversal starts at the root, and bits are read from the bitstream to pick the first or second child of the current node until a leaf node is reached. The data value of the output is the value of this leaf […] Trie nodes that are not leafs. use 1 0-value bit. Trie nodes that are leafs use 1 1-value bit and a 16-bit data value. length is the number of 16-bit values to decode. Each value normally requires >= 1 bit from the bitstream to resolve. This algorithm is vulnerable if the root node of the trie is a leaf, i.e., the trie only has a single value. In this case, the value requires 0 bits from the bitstream to resolve. The output file will consist of the value stored by the root node repeated length times. The maximum value of length is 2147483647, so it's possible to cause the remote side to create a $2147483647 * 2$ B file with a Spiffy archive that's only 11 B large."

4.4.2.3.3 Raytheon/BBN Technologies

BBN provided 8 challenge programs for E4.

Table 23: Engagement 4 Raytheon/BBN Technologies Programs and Questions

Challenge Program	# of Intended Positive Qs	# of Intended Null Qs	Total # of Qs	# of Attempts	# of Correct Attempts*	Percentage Correct
Collab	0	3	3	14	9	64%
Info Trader	1	1	2	5	1	20%
Linear Algebra	2	0	2	10	8	80%
Malware Analyzer	1	1	2	14	4	29%
RSA Commander	1	1	2	8	3	38%
SmartMail	1	2	3	11	7	64%
Tour Planner	1	2	3	14	9	64%
Tweeter	2	0	2	10	8	80%
Total	9	10	19	86	49	57%

4.4.2.3.4 CyberPoint

CyberPoint provided 17 challenge programs for E4.

Table 24: Engagement 4 CyberPoint Programs and Questions

Challenge Program	# of Intended Positive Qs	# of Intended Null Qs	Total # of Qs	# of Attempts	# of Correct Attempts	Percentage Correct
Airplan 1	1	1	2	9	7	78%
Airplan 2	2	0	2	14	13	93%
Airplan 3	2	1	3	18	15	83%
Airplan 4	1	1	2	12	11	92%
Airplan 5	1	1	2	9	8	89%
BidPal 1	0	2	2	8	4	50%
BidPal 2	2	0	2	9	6	67%
PowerBroker 1	2	0	2	11	9	82%
PowerBroker 2	0	2	2	11	7	64%
PowerBroker 3	0	2	2	2	2	100%
PowerBroker 4	1	1	2	7	6	86%
WithMi 1	2	1	3	12	8	67%
WithMi 2	1	1	2	7	4	57%
WithMi 3	1	1	2	9	9	100%
WithMi 4	3	1	4	21	16	76%
WithMi 5	0	4	4	18	12	67%
WithMi 6	0	2	2	5	2	40%
Total	19	21	40	182	139	76%

4.5 Engagement 5

4.5.1 Engagement Plan

4.5.1.1 Purpose and Program Scope

4.5.1.1.1 Program Scope

This Engagement Plan is for Engagement 5 which is both a take-home and live collaborative engagement. Take-home engagement answers must be submitted by noon Eastern time on January 26th, 2018. The live collaborative engagement will take place from February 13th – February 15th, 2018; answers must be submitted at the end of the live collaborative engagement on February 15th, 2018. Answers should be sent to evan.fortunato@apogee-research.com and kwame.ampeh@apogee-research.com or uploaded to Apogee's Gitlab server.

The document is structured into the following sections.

Section 1: Purpose and Program Scope
Section 2: Operational Definitions and Research Hypotheses
Section 3: Engagement 3 Plans and Logistics
Section 4: Post-Engagement Analysis Plans

4.5.1.2 Operational Definitions and Program Hypotheses

4.5.1.2.1 Operational Definitions of STAC Vulnerabilities

To guide the Engagements, the STAC AC, EL, and government teams have come up with operational definitions of vulnerability to algorithmic complexity attack and to side channels. The AC performers will use these definitions to determine when the vulnerabilities they've inserted into their challenge programs are strong enough for use in engagements. The Control and R&D Teams will use these operational definitions to determine when they've found a vulnerability worth reporting – i.e., if it meets the definition of a STAC vulnerability. The operational definition document, currently *2017-08-22-opdef-v08.1*, specifies the definitions for STAC vulnerabilities along with guidance for Blue team responses to engagement challenges.

4.5.1.2.1.1 Unintended Vulnerabilities in Challenge Programs

Many Algorithmic Complexity and Side Channel vulnerabilities may exist in each challenge program beyond those intentionally inserted. For example, a normal authentication function is technically a side channel that leaks a secret. However, most of these vulnerabilities are "weak" – e.g., for Algorithmic Complexity, the additional complexity is a small percentage of the normal complexity, and for Side Channels, the number of operations necessary to resolve the secret is impractically large.

The defined operational definitions defined above are designed to minimize these unintended vulnerabilities in the challenge programs with the use of strong vulnerability thresholds (e.g., adversary operation budget is small for Side Channels). These thresholds will serve to discount most of these "unintended" vulnerabilities as not meeting the STAC definition. However, some unintended vulnerabilities will occur that do meet the definition. A reported actual vulnerability that satisfies the operational definition criterion for a given challenge program is a correct

response, regardless if it is an intended or unintended vulnerability on behalf of the AC teams. During the analysis period after each Engagement, the EL team will work with the R&D and Control performers to verify that reported unintended vulnerabilities do actually meet the operational definition so that full credit is given. Unintended vulnerabilities can fall into two categories: those "in-scope" as per the STAC Operational Definitions document and those "out-of-scope" as per the STAC Operational Definitions document. Full credit will be given for all verified unintended vulnerability whether they are "in-scope" or "out-of-scope." In the case of "in-scope" unintended vulnerabilities, the engagement answer key will be updated to reflect the identified unintended vulnerabilities.

4.5.1.2.2 Research Hypothesis and Segmentation of Challenge Problems

Throughout the STAC program, the EL team will have a set of research hypotheses pertaining to performance of the tools with respect to different types of STAC vulnerabilities and challenge problems. Example hypotheses include:

13. Complexity vulnerabilities can be due to: (1) a change in the complexity of the algorithm, (2) a change in the coefficients of a fixed complexity, or (3) a flawed data guard that allows processing of input data that shouldn't be allowed. How do the different R&D techniques perform with respect to each of these types of complexity vulnerabilities?
14. Side channels can be weakened by increasing the number of secret symbols that are mapped to identical observed symbols or by increasing the width of the parent distributions of the observed symbols (causing overlaps of the of the samples from different observed symbol distributions). How do the different R&D techniques perform with respect to each of these challenges?
15. Some non-vulnerable programs are locally balanced (e.g., all paths through a sub-section of the binary consume equivalent time and space resources) while others are non-locally balanced (e.g., resource consumption is asymmetric on a set of paths in multiple parts of the program resulting in entire execution traces that are balanced). How do the different R&D techniques perform with respect to the locality of balancing of resource consumption?

In support of these hypotheses, each challenge problem will be assessed and categorized with respect to how results on the challenge problem will support resolution of each of the hypotheses. This segmentation of the challenge problems will not be exposed during the engagements, but will be tracked for post-engagement analysis.

During the engagements, each challenge problem will come questions such as:

17) Does the challenge problem contain a time-based side channel?
18) Does the challenge problem contain a space-based side channel?
19) Does the challenge problem contain a time-based complexity vulnerability?
20) Does the challenge problem contain a space-based complexity vulnerability?

Note, that the questions will be more specific version of these classes of questions including additional context information and could include questions regarding combined time and space vulnerabilities.

4.5.1.3 Engagement 5 Details and Logistics

4.5.1.3.1.1 Engagement 5 Details

Engagement 5 differs from previous engagements in that it is a combined take-home and live engagement. The take-home portion of the engagement will take place from 1/3/2018 – 1/26/2018. Following the submission of the take-home engagement responses, the EL will use the results to determine the **collaborative groups** (teams that can collaborate) for each challenge question. Each challenge question with non-uniform responses (the same response is not provided by all teams that responded) will be available for collaboration. The **collaborative group** for each collaborative challenge question will be comprised of the teams that submit a response for that question during the take-home engagement. The list of **collaborative groups** for each challenge question will be provided to Blue Teams at the PI meeting. Following a collaboration, teams are not required to reach a consensus on a response. Only R&D Teams will participate in the live collaborative portion of Engagement 5.

Unlike for previous engagements, the PI meeting (2/12/2018) will take place before the live collaborative engagement (2/13/2018 – 2/15/2018). The purpose of this PI meeting is for teams to pitch the capabilities of their tools to other teams; highlighting the strengths and noting the weaknesses in their tools. To this end, each team should identify 3 challenge questions they are confident in that demonstrate the breadth of their approach and 3 challenge questions that they are not confident in, specifying why their approach struggles with these questions. Previous PI meeting presentations were used to provided DARPA updates on research and tool development. Teams will be asked to provide this information to DARPA prior to the PI meeting and to focus on using this February PI meeting to demonstrate where their tools excel and where their tools struggle.

4.5.1.3.1.2 Engagement 5 Schedule

The schedule below provides an overview of the milestones for Engagement 5, including the engagement itself and the post engagement evaluation.

- Take-home Engagement 5 (all STAC teams): Event runs from 1/3/2018 – 1/26/2018.
- PI Meeting will take place on 2/12/2018.
- Live collaborative Engagement 5 (STAC R&D Teams): Event runs from 2/13/2018 – 2/15/2018.
- Release Engagement Plans (EL team): Initial release on 9/19/2018. Post on GitLab Public_EL_Information Project.
- AC Teams deliver completed versions of all engagement challenges for review to the EL: 10/02/2017 by noon eastern time.
- AC Teams deliver final versions of all Engagement 5 challenges to the EL: 11/16/2017 by noon eastern time.
- The EL team will release the take-home Engagement 5 challenges to the Blue Teams on 1/3/2018.
- BTs submit take-home Engagement 5 responses to the EL: 1/26/2018 by noon eastern time.
- Live collaborative Engagement 5 will take place at DARPA from 2/13/2018 to 2/15/2018.
- Quick-look Engagement 5 results (EL team): initial Engagement 5 quick-look results will be presented after the conclusion of the live collaborative engagement on 2/15/2018. Reviewed results will be subsequently posted to each team's repo on GitLab by 3/15/2018.

- Teams will have until 4/13/2018 to request evaluation clarifications and changes.
- Final Engagement 5 analysis report delivered to DARPA: 5/15/2018.

4.5.1.4 Post Engagement Analysis Plans

4.5.1.4.1.1 Metrics

We will be using the following set of metrics for Engagement 5:

- True Positive Rate (Number of correctly identified programs as vulnerable / Total number of vulnerable programs)
- True Negative Rate (Number of correctly identified programs as not vulnerable / Total number of non-vulnerable programs).

Note, True Positive Rate is equivalent to a Probability of Detection Metric while the True Negative Rate is equal to 1 – False Alarm Rate. Using the True Positive and Negative Rate gives insight into the relative strengths of each of the teams with respect to finding and ruling out vulnerabilities but doesn't give us insight into the speed of the teams.

To account for the speed, we will consider two different segmentations of the data. In the first segmentation, only properly justified answers will be considered (in the numerator) and only answered challenge questions will be considered (in the denominator). This will allow us to see the performance of each approach on the set of problems for which they chose to answer. In the second segmentation, properly justified answers will be considered (in the numerator) and all challenge questions will be considered (in the denominator). This second segmentation will show the benefits of speed as teams that only answer a small fraction of the questions will not be able to score well under the second segmentation. As teams should answer most of the questions in Engagement 5, the segmentation against the set of all challenge questions will be weighted more in evaluating Blue Team performance than in previous engagements.

To achieve the segmentation, each team's answers will be evaluated against the intended vulnerabilities of the challenge programs. In cases where a team believes that there is a vulnerability when none was intended, the EL team will re-evaluate the challenge application (using the justification provided by the team) to determine if an unintended vulnerability is present in the challenge application. If there is an unintended vulnerability, the EL will determine there is an in-scope unintended vulnerability. If there is an in-scope unintended vulnerability, then the challenge program status will be updated to act as though that unintended vulnerability was intended, and all teams' answers will be rescored against this updated status. If there is an unintended out-of-scope vulnerability, then the status of the challenge program will not be updated (so other teams' scores will not change) but we will score the team that found the unintended out-of-scope vulnerability as if the challenge program in question is vulnerable. Note, this means that the denominators of the metrics for each team will be different even under the second data segmentation.

Next, for each team we will evaluate the justifications against the justification guidance provided. If the justification is sufficient, then the answer will be considered a justified answer and will be included in the justified answer analysis. Independent of the validity of the justification, if a question is answered, then the challenge program will be included in the denominator under the first data segmentation.

The performances of Blue Teams during the take-home and live collaborative (only R&D Teams) portions of Engagement 5 will be compared to evaluate the impact of the collaborative engagement. To determine the impact of each R&D Team on their collaborators, we will calculate the total score change (before vs after the collaborative engagement) for each team's collaborators.

Finally, for the take-home portion of Engagement 5, we ask each team to track and report the total hours spent on each challenge question. This information will not be used to scale scores; instead, it will be used to gain a better understanding of which types of questions are easier and which are more difficult for each team.

4.5.1.4.1.2 Initial Assessments, Refinements Based on Blue Team Feedback and Site Visits

After lunch on the final day of the live collaborative engagement (2/15/2018), the EL team will provide an initial assessment of the performance of each team against the program metrics using the intended vulnerabilities (i.e., those vulnerabilities that were intentionally added to the challenge programs). Following the release of individual assessments to each team on 3/15/2018, teams will have until 4/13/2018 to request evaluation clarifications and changes.

4.5.2 Engagement Results

4.5.2.1 Overview

STAC Engagement 5 took place from January 3rd, 2018 to January 26th, 2018 and from February 14th, 2018 to February 16th, 2018. The engagement incorporated changes to the STAC operational definitions and reference platform that improved upon previous iterations and created a cleaner platform for experimentation. In addition, Engagement 5 departed from the engagement format used in Engagements 1 through 4. Engagements 1 through 4 alternated between live engagements and 3-month take-home engagements: engagements 1 and 3 were live engagements and engagements 2 and 4 were take-home engagements. During the live engagements, teams arrived at a central location, received the challenge programs, and were given two 8-hour days to work on the challenges. The 3-month take-home engagement would begin shortly after the live engagement and teams would have the same challenge programs from the live engagements with most of the unintended vulnerabilities identified during the live engagement removed. Some Blue Teams with more lightweight tools performed better in the live engagements while others with more computationally intensive tools would perform better in the take-home engagements. In Engagement 5, the live/take-home format was replaced with a short take-home component and a live collaborative component combined into a single engagement. The short take-home component allows both the computationally intensive and more light-weight tools sufficient time to demonstrate their capabilities. The live-collaborative component allows Blue Teams the opportunity to better understand other teams' approaches to STAC and allows DARPA to see the combined capability of the Blue Teams to solve STAC challenge problems. Finally, Engagement 5 was the first engagement to contain an uneven distribution of vulnerable and non-vulnerable challenge questions. The uneven distribution of challenge questions was intended to test how well the teams rely on their tools. The results from Engagement 5 combined with those from Engagement 6, which will also contain a skewed challenge question distribution but favoring non-vulnerable challenge questions, should help to push for more tool reliance.

The performance of Blue Teams in Engagement 5 provided additional insight into the current capabilities of their tools. Collectively, the teams demonstrated a strong ability to evaluate

Algorithmic Complexity (AC) challenge questions and had more difficulty with the Side Channel (SC) challenge questions, specifically SC Time questions. Secondly, Blue Teams demonstrated that collectively they can analyze localized behavior but struggle to analyze non-localized behavior.

In Engagement 5 (E5), 31 questions were asked about 15 different challenge programs. Of the 25 challenge programs, 6 were provided by CyberPoint and 9 were provided by Raytheon/BBN Technologies (BBN). The 31 questions can be broken into 10 different categories. 5 different question types were asked (Algorithmic Complexity (AC) in Time, AC in Space, Side Channel (SC) in Time, SC in Space, and SC in Space and Time). Table 15 provides the breakdown of the 31 questions asked.

Table 25: How many questions were asked for a given question type and intended question result

Question Type	Question Result	Number of Questions
Algorithmic Complexity in Time	Positive	6
Algorithmic Complexity in Space	Positive	8
Side Channel in Time	Positive	4
Side Channel in Space	Positive	7
Side Channel in Time and Space	Positive	0
Algorithmic Complexity in Time	Null	0
Algorithmic Complexity in Space	Null	1
Side Channel in Time	Null	2
Side Channel in Space	Null	2
Side Channel in Time and Space	Null	1

Eight R&D Teams and one control team, collectively known as the Blue Teams, participated in the engagement. In total, the Blue Teams attempted 207 questions during the take-home engagement and 217 questions during the live-collaborative engagement (with 179 and 189 of those attempted by the R&D Teams during the take-home and live-collaborative components of the engagement respectively). The total combined accuracy of the Blue Teams was 57% for the take-home component and 89% for the live component (55% and 89% for just the R&D Teams in the take-home and live components respectively). Table 26 provides a listing of the teams that participated in Engagement 5.

Table 26: The different teams that participated in Engagement 5

Blue Teams	Research Teams	University of Colorado Boulder (UC Boulder)
		Draper Labs
		GrammaTech
		Iowa State University (ISU)
		Northeastern University (NEU)
		University of Maryland (UMD)
		University of Utah
		Vanderbilt University
	Control Team	Two Six Labs
Red Teams		CyberPoint
		BBN

A breakdown of the number of questions answered for each category and the resulting accuracy following a review by the EL is shown in Table 27.

Table 27: The distribution of Blue Team responses across different categories in the take-home component of E5

	Total Responses	Correct Reviewed Responses	Accuracy
SC Space	58	32	55%
SC Time	42	24	57%
SC Space/Time	4	2	50%
AC Space	60	33	55%
AC Time	43	27	63%

Table 28: The distribution of Blue Team responses across different categories in the live-collaborative component of E5

	Total Responses	Correct Reviewed Responses	Accuracy
SC Space	58	47	81%
SC Time	45	36	80%
SC Space/Time	4	4	100%
AC Space	64	62	97%
AC Time	46	45	98%

In the take-home component of Engagement 5, the Blue Teams had the highest accuracy for AC Time questions with a 63% accuracy. The teams' performance across the other categories ranged from 50% to 57%. In contrast, following the live-collaborative component of the engagement the Blue teams' collective accuracy in all categories was greater than 80%. In future engagements, the intent is to continue to create more challenging non-localized AC and SC challenges.

4.5.2.1.1 Take-Home Engagement

This section evaluates Blue Team performance on a set of metrics measuring overall accuracy and performance on vulnerable versus non-vulnerable challenge questions, while accounting for the number of responses provided. These metrics applied to the take-home component of Engagement 5 show that the control team outperformed the R&D Teams, with Colorado leading the performance of the R&D Teams. These metrics don't provide a complete picture of team performance; we must also consider a combination of these metrics and the quality of the responses provided by the teams coupled with an understanding of their research approaches. When these are considered Colorado, Vanderbilt, GrammaTech and Two Six Labs are in the first tier; Iowa, Northeastern, and Utah in the second tier; and UMD and Draper are in the third tier.

Engagement 5 contained a distribution of intended vulnerable and intended non-vulnerable challenge programs. During the engagement, Blue Teams may discover unintended vulnerabilities or discover that an intended vulnerability is not sufficiently strong. These discoveries are classified as *in-scope* if they satisfy the STAC operational definition

requirements, and *out-of-scope* if they do not. *In-scope* discoveries result in changes to the engagement answer key for all teams. *Out-of-scope* discoveries only result in changes to the engagement answer key for the team that made the discovery. The engagement response review process takes the justification into account in evaluating the binary (vulnerable/non-vulnerable) responses. The figures below show the reviewed accuracies for Engagement 5.

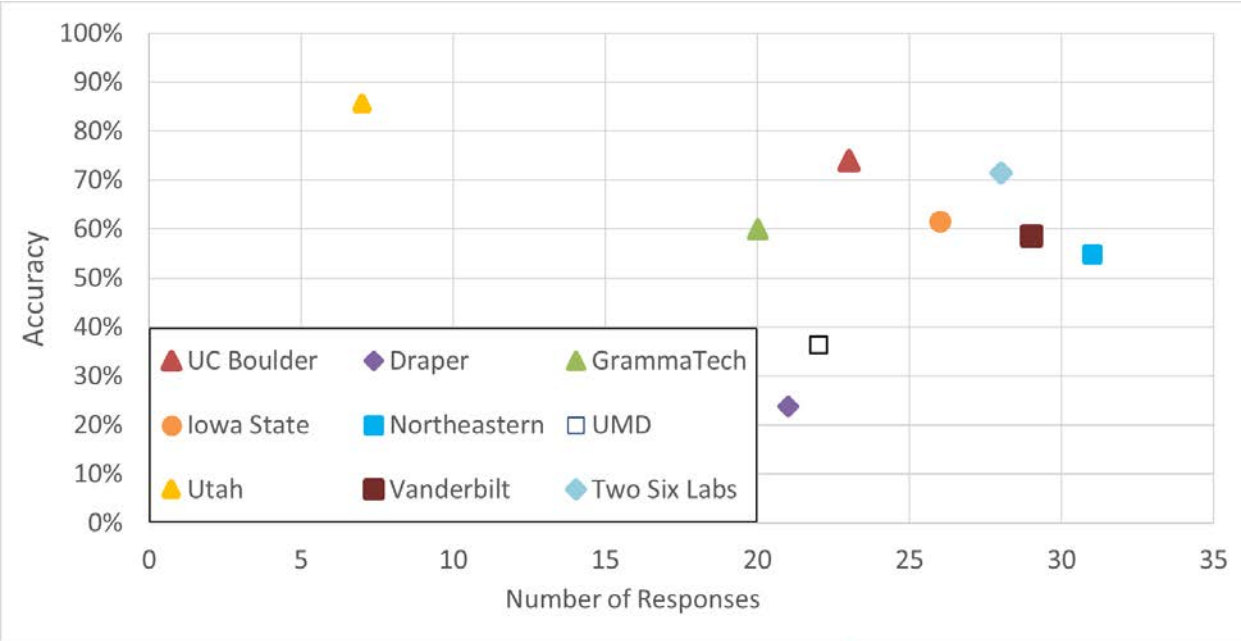


Figure 47: E5 Take-home reviewed accuracy versus number of responses.

The plot of the take-home reviewed accuracy versus number of responses shows that Utah had the highest take-home accuracy of 86% but only answered 7 questions. In contrast, the Blue Team with the next highest accuracy, Colorado had a 74% accuracy and answered three times as many questions. UMD and Draper had the lowest performance with accuracies of 36% and 24% respectively.

In Engagement 5, the guidance provided to Blue Teams was to attempt to answer *most* of the questions. Figure 47 compares Blue Team accuracies on answered questions to their accuracies on all questions. Against the segmentation of all questions, the control team (Two Six Labs) had the highest accuracy; the other teams in the first performance tier were Vanderbilt, Northeastern and Colorado all with a greater than 50% accuracy against the set of all questions. The second tier of 30% to 50% accuracy comprises Iowa, and GrammaTech; while the third tier of below 30% accuracy comprises UMD, Utah, and Draper. Against the set of all questions, the Blue Teams were outperformed by the control team, with the control team being the only team to have an accuracy greater than 60%. As this engagement contained significantly more vulnerable than non-vulnerable challenge questions, this may have played more to the capability of the control team to identify vulnerabilities. Engagement 6, where the distribution of questions is skewed in favor of non-vulnerable questions, should provide a comparative picture of the R&D Teams versus the control team in ruling out vulnerabilities.

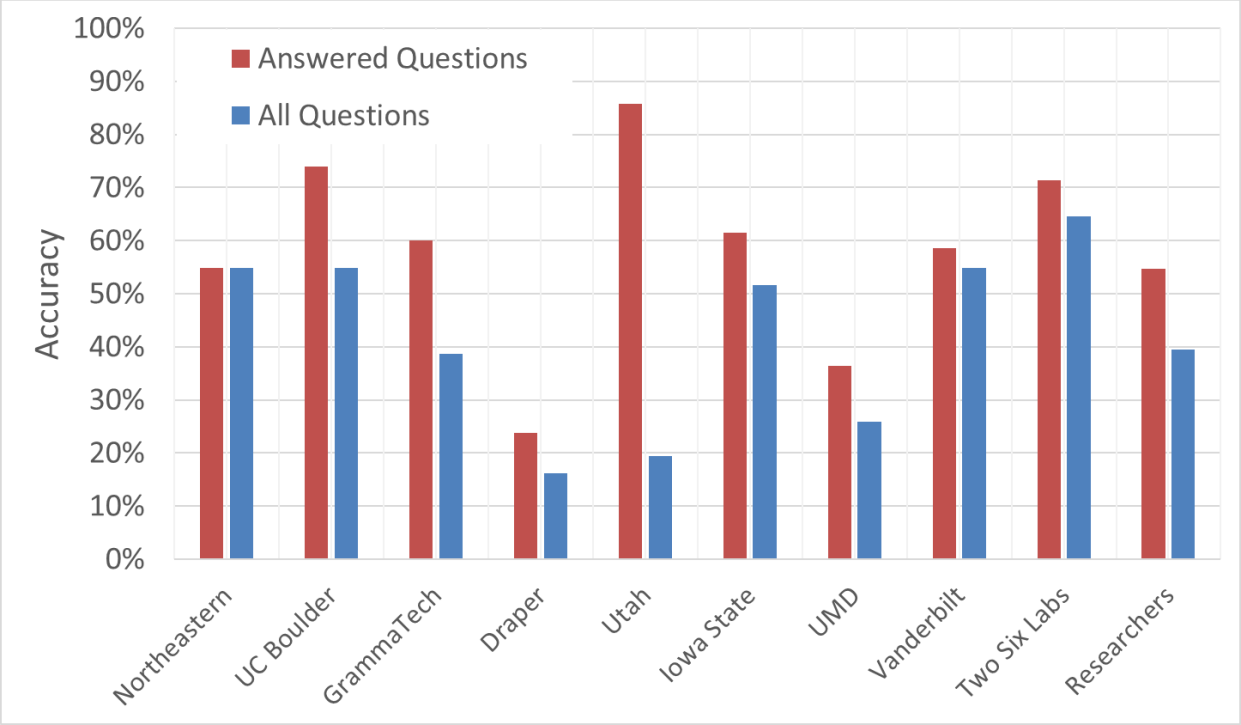


Figure 48: E5 Take-home reviewed accuracy against the set of answered questions compared to accuracy against the set of all questions.

STAC engagements test Blue Team ability to identify vulnerabilities but place more focus on the ability to rule out vulnerabilities with sufficiently high accuracy. The metric of total accuracy fails to capture how teams perform against vulnerable versus non-vulnerable challenge questions. As with prior engagements we computed the true positive rate (TPR) as a measure of a team's ability to identify vulnerabilities and the true negative rate (TNR) as a measure of a team's ability to rule out vulnerabilities. The TPR is the number of correct "Yes" responses divided by the number of vulnerable challenge questions. The TNR is the number of correct "No" responses divided by the number of non-vulnerable challenge questions.

A ROC-curve style plot of TPR versus TNR is used to visualize the relative performances of the different Blue Teams. First, we look at Blue Team performance against the set of questions answered by each team. As shown in Figure 49, in this segmentation, the highest performance tier comprises Utah, Two Six Labs, and GrammaTech. Colorado and Vanderbilt compose the upper middle tier, Iowa and Northeastern compose the lower middle tier, and the lowest tier is made up of UMD and Draper.

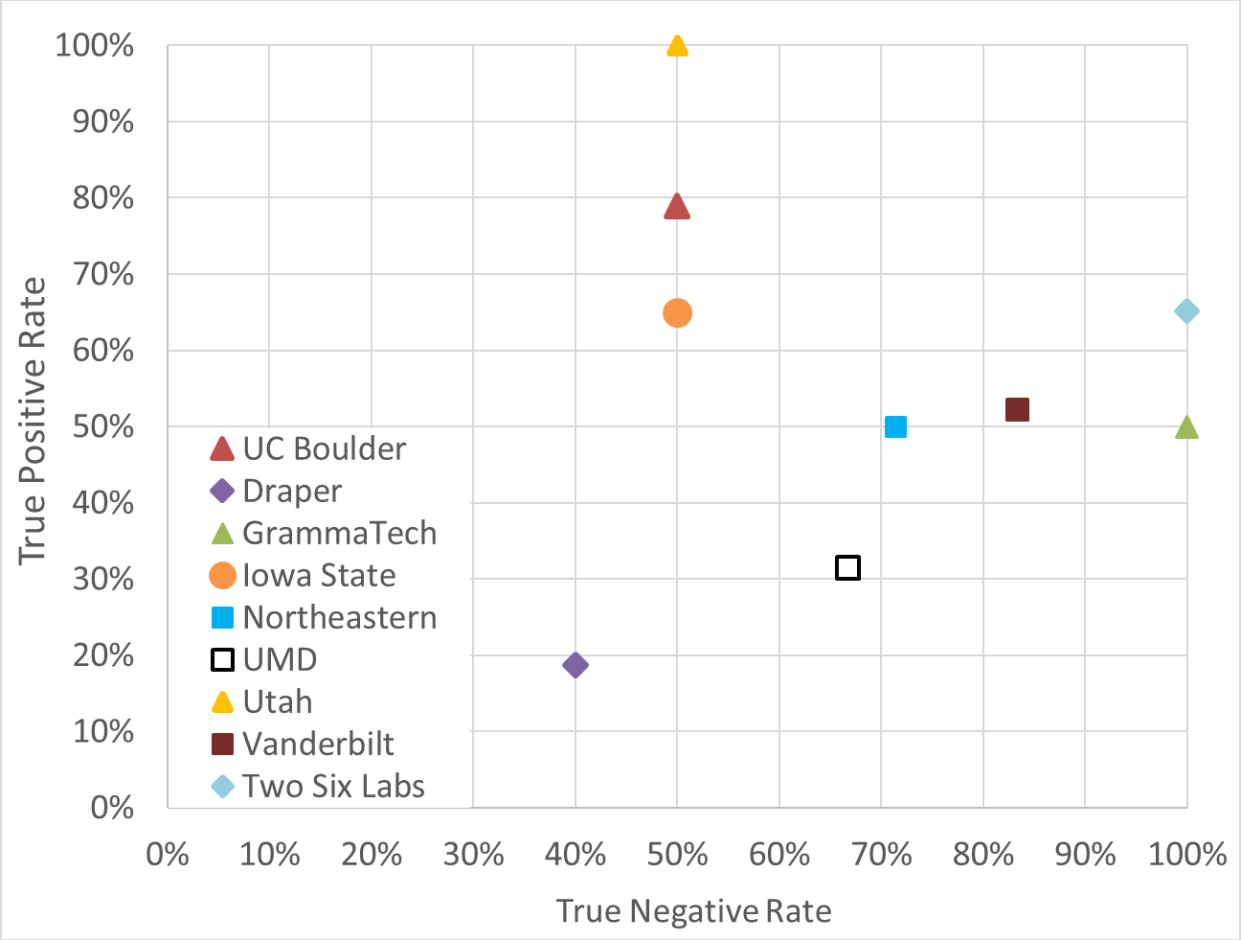


Figure 49: E5 Take-home TPR versus TNR segmentation of answered questions.

The segmentation of answered questions does not account for the percentage of total questions answered by a Blue Team. To account for this, we use the same metrics of TPR and TNR but treat unanswered questions as incorrect. The results, shown in Figure 50 paint a different picture. Against the segmentation of all questions, only Two Six Labs, Northeastern and Vanderbilt have a greater than or equal to 50% TPR and TNR. These three teams make up the first performance tier. The second tier comprises Colorado, Iowa, and GrammaTech and the third tier comprises Utah, UMD, and Draper.

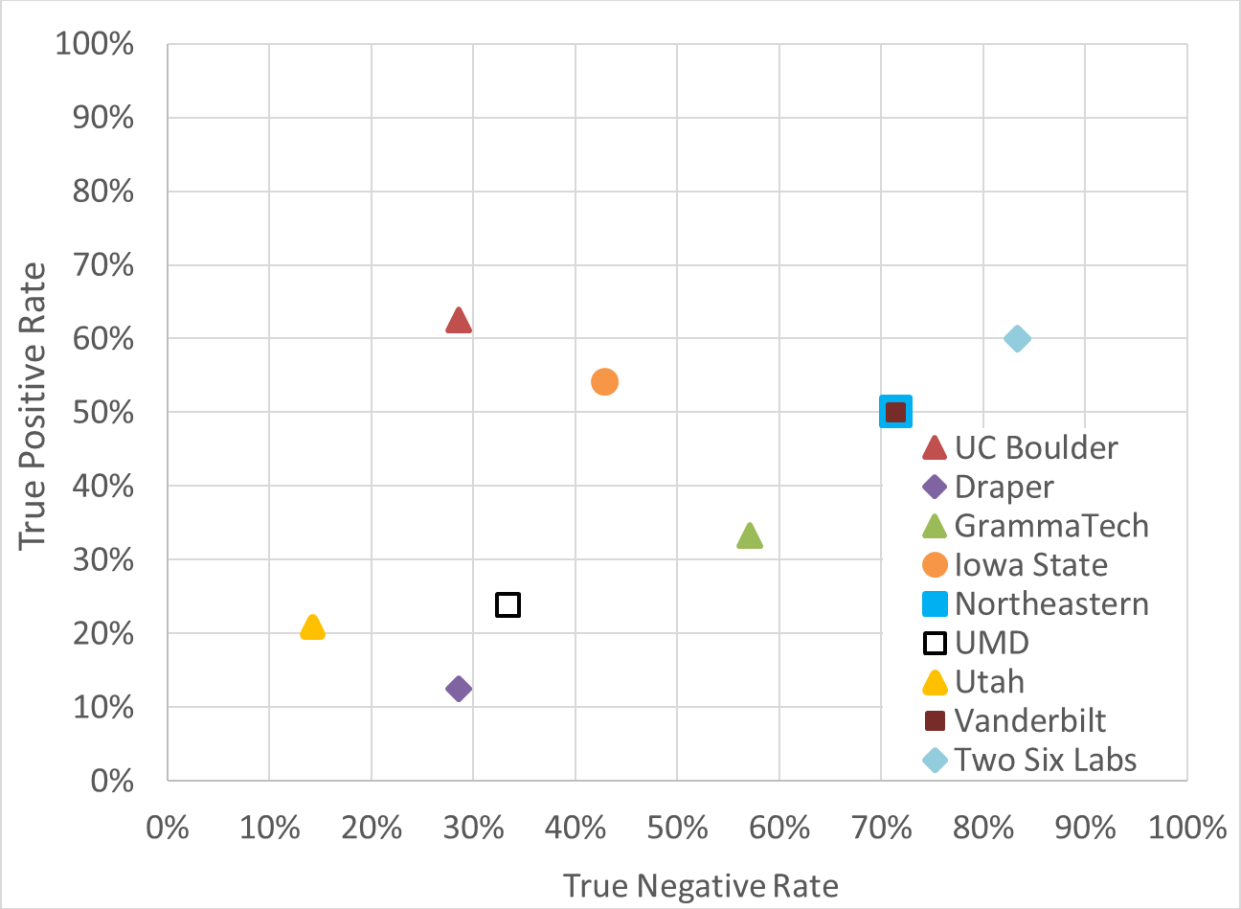


Figure 50: E5 Take-home TPR versus TNR segmentation of all questions.

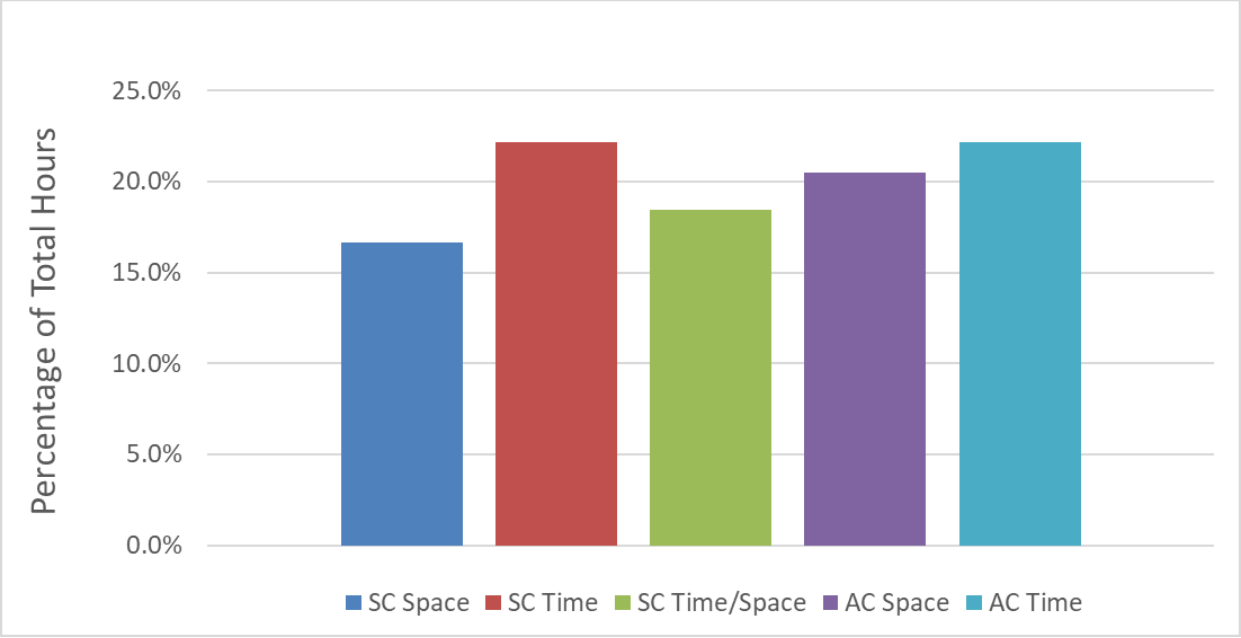


Figure 51: E5 Take-home normalized percentage total time spent per challenge question category

During the take-home engagement, Blue Teams were asked to record the total number of hours spent on each challenge question. The resulting data on the amount of time spent on each challenge question should provide a relative measure of the difficulty of different types of challenge questions. The resulting data was scaled by the total number of challenge questions answered for each category as the categories did not have an equal number of responses. The scaled number of hours data shows that Blue Teams spent the most time on AC Time (22% of total time) and AC Space (22% of total time) questions. Teams spend the least time of SC Space questions (17% of total time). Segmented by AC Team, Blue Teams spent approximately 45 minutes more per BBN challenge question than CyberPoint challenge question; however, the time spent per BBN challenge question was only 7% more than on CyberPoint question.

4.5.2.1.2 Live-Collaborative Engagement

The live-collaborative component of Engagement 5 allowed Blue Teams to collaborate on their take-home engagement responses. Teams could work on questions they answered during the take-home with others who answered the same questions during the take-home. Blue Teams were encouraged to focus on the questions answered during the take-home but once completed, could work on questions not answered during the take-home engagement with other teams who did not answer those questions during the take-home engagement. Engagement 5 contained 31 challenge questions; therefore, to help teams prioritize, a heuristic measure was used to rank the collaborative questions. This heuristic prioritizes questions based on the potential for teams to disagree following collaboration and the potential for teams to change their responses following collaboration.

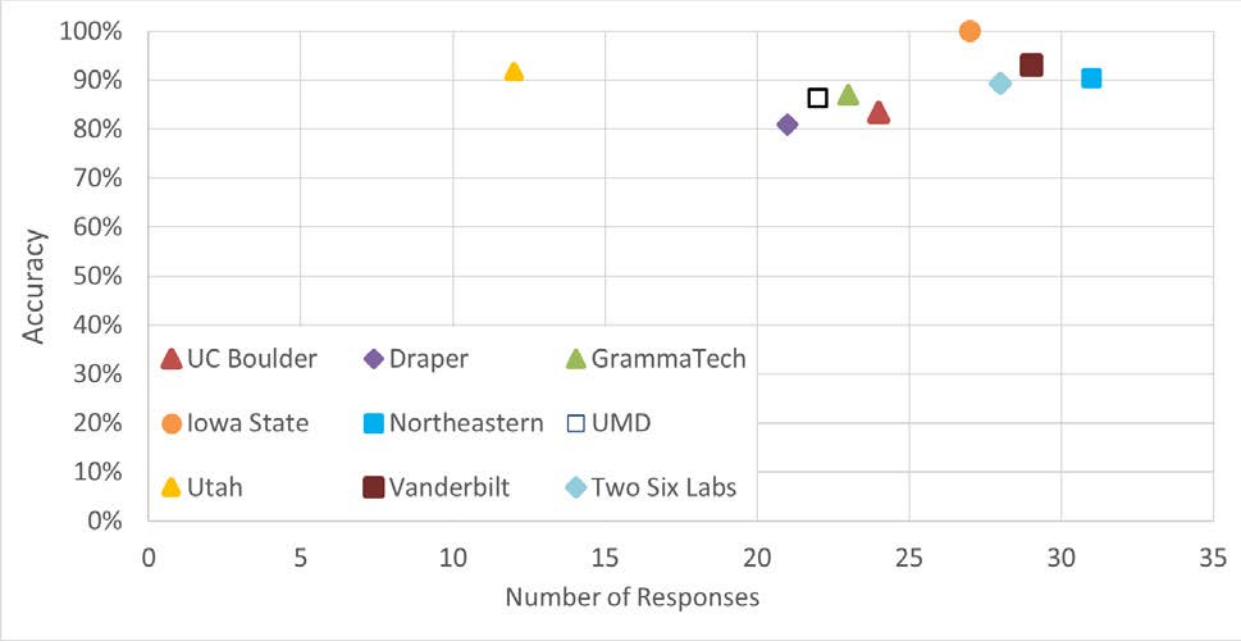


Figure 52: E5 Live-collaborative reviewed accuracy versus number of responses.

The results of the live-collaborative engagement, accounting for both the questions answered during the take-home engagement and those answered for the first time at the live-collaborative engagement, are shown below. First, we look at overall accuracy against the segmentation of answered questions shown in Figure 52.

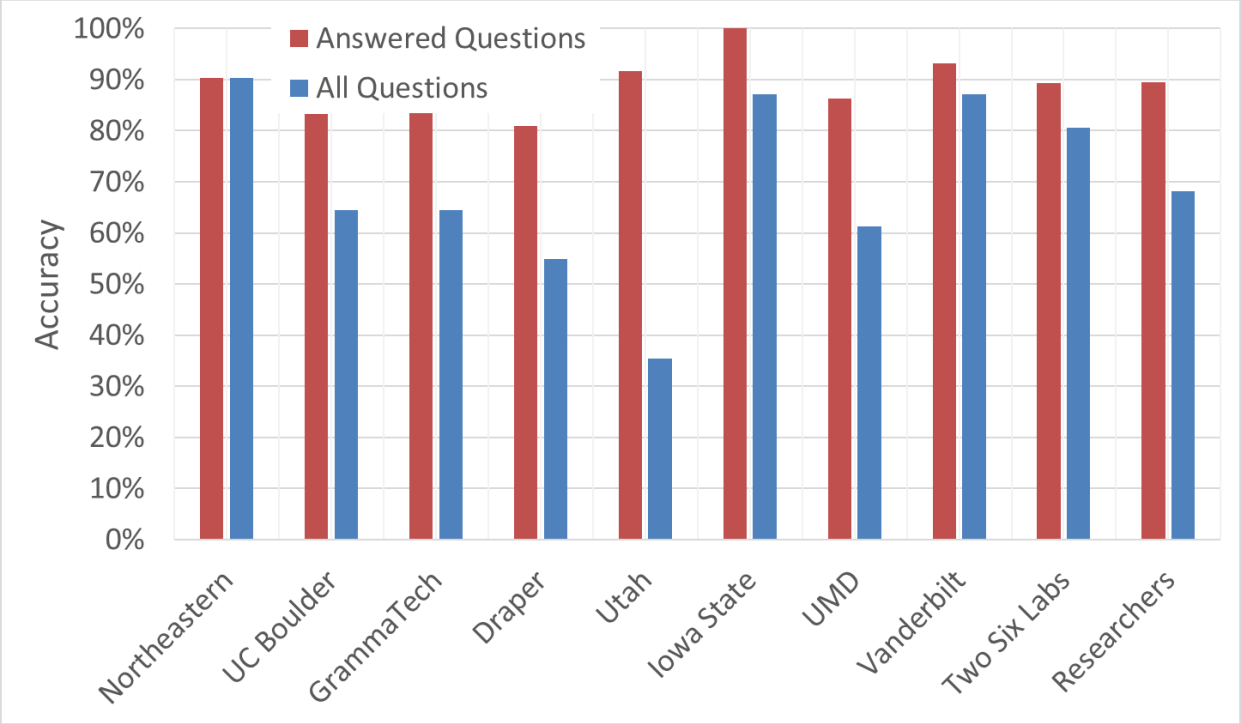


Figure 53: E5 Live-collaborative reviewed accuracy against the set of answered questions compared to accuracy against the set of all questions.

Figure 52 shows the reviewed accuracy versus the number of responses. Following the live-collaborative component of the engagement all teams increased their accuracy from the take-home engagement. Iowa State achieved a 100% post-collaborative engagement accuracy on 27 questions with all teams achieving an accuracy of greater than 80%.

Comparing against the set of all questions (Figure 53), Northeastern had the highest accuracy with 90%; three other teams, Iowa, Vanderbilt, and Two Six Labs achieved a greater than 80% accuracy. Draper and Utah were the only teams with below 60% accuracy in this segmentation.

Iowa led the way in total accuracy with a 100% accuracy against the set of 27 answered questions. This is reduced to 87% when counted against the set of all 31 questions.

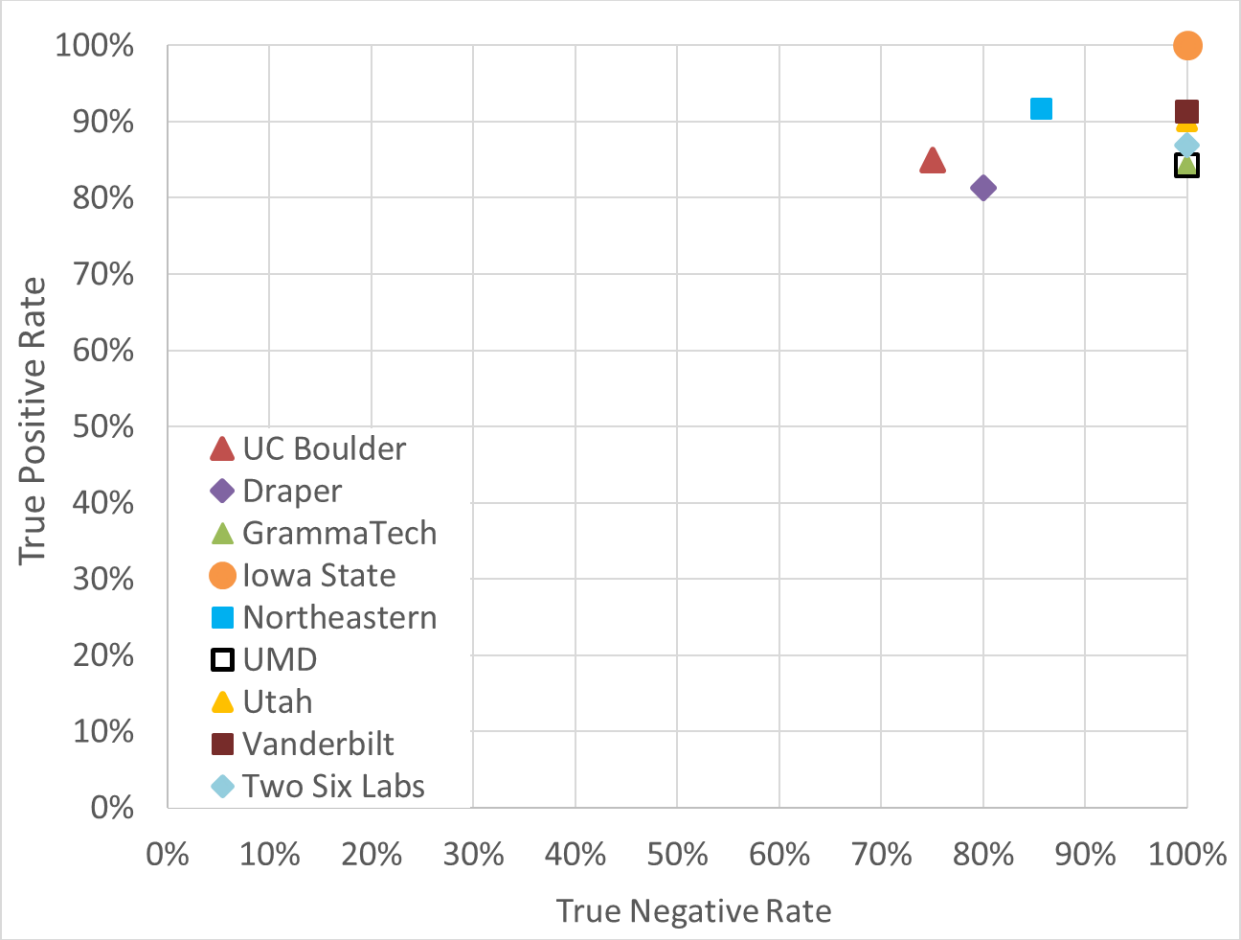


Figure 54: E5 Live-collaborative TPR versus TNR segmentation of answered questions.

Figure 54 shows the TPR versus TNR performance against the set of answered questions. Iowa's 100% accuracy results in a 100% TPR and TNR. Vanderbilt, Utah, Two Six Labs, GrammaTech, and UMD are all closely stacked at 100% TNR and between 87% and 91% TPR. Colorado, Northeastern and Draper remain outside this primary cluster of teams.

Against the segmentation of all questions (Figure 55) there are three distinct groupings of teams. The highest performance tier comprises Northeastern, Iowa State, Vanderbilt, and Two Six Labs. The middle tier comprises Colorado, UMD, Draper, and GrammaTech; with Utah making up the last tier due to their limited set of responses.

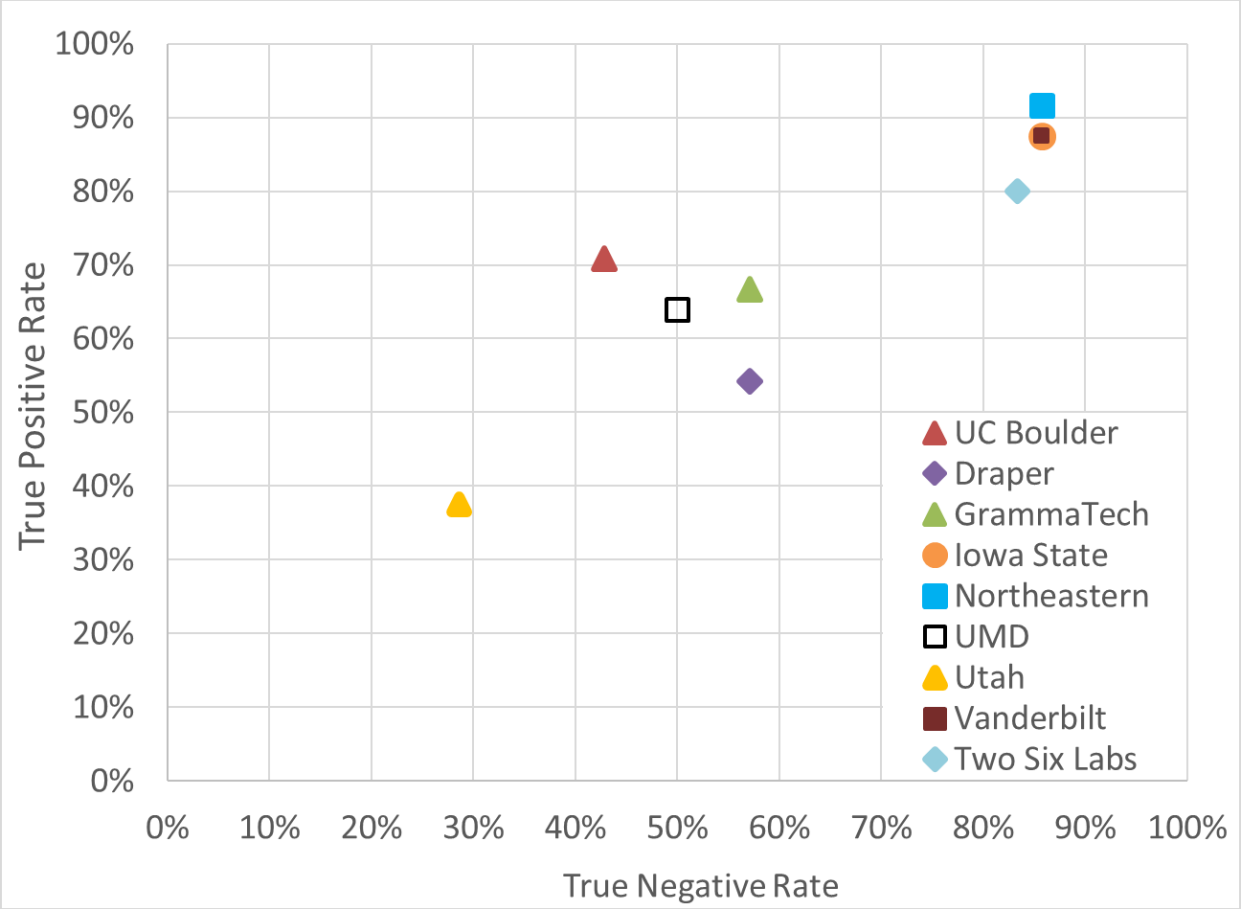


Figure 55: E5 Live-collaborative TPR versus TNR segmentation of all questions.

To highlight Blue Team performances in the live-collaborative engagement, Figure 56 shows the accuracies against the set of answered questions for different segmentations of Engagement 5 and Figure 57 shows the percentage point difference between the take-home and live-collaborative engagement components.

In Figure 56, in cases where teams changed their answer following collaboration, all teams achieved a 100% accuracy. GrammaTech was the only team that improved their accuracy by answering questions in the live-collaborative engagement that they failed to answer during the take-home engagement.

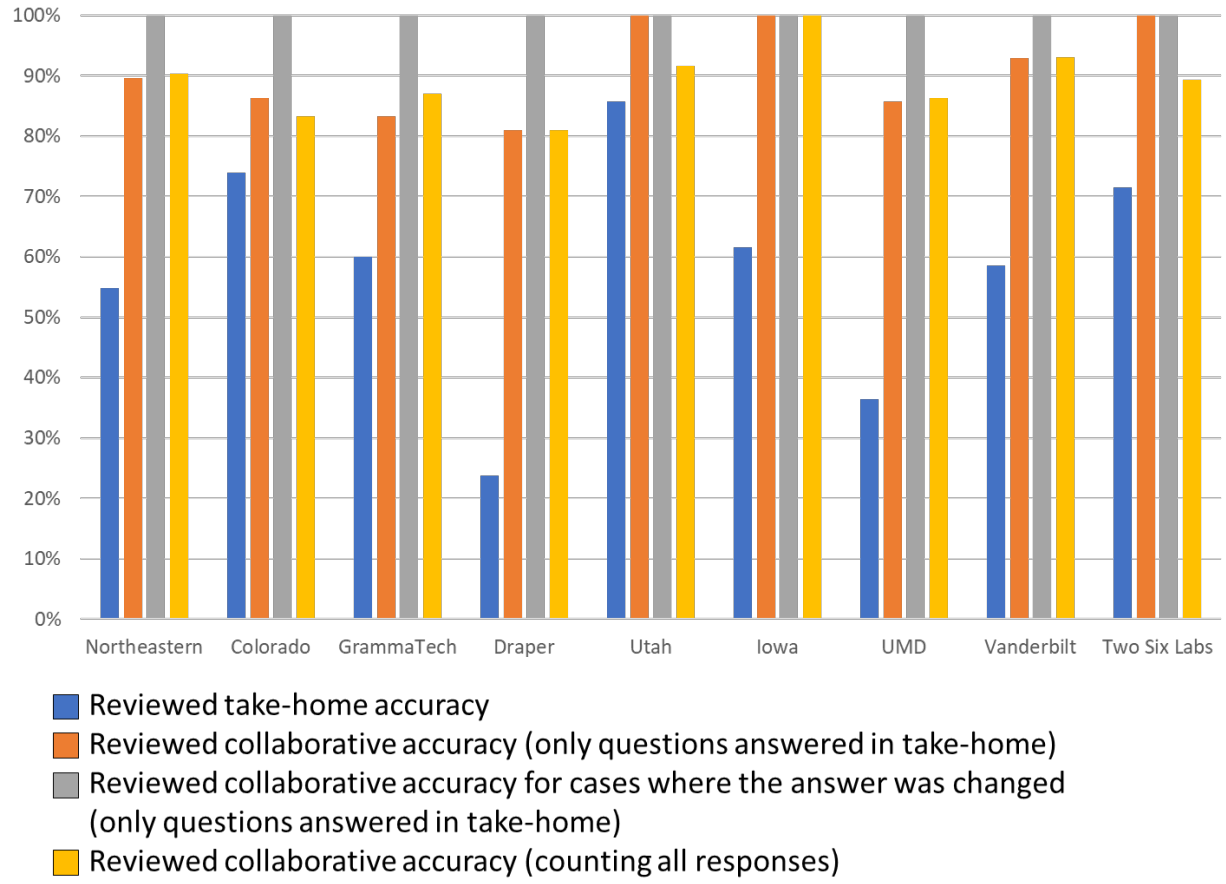


Figure 56: E5 Take-home and live-collaborative accuracies

Figure 57 shows that Draper, UMD, and Iowa had the greatest percentage point increases from the take-home component to the live-collaborative component.

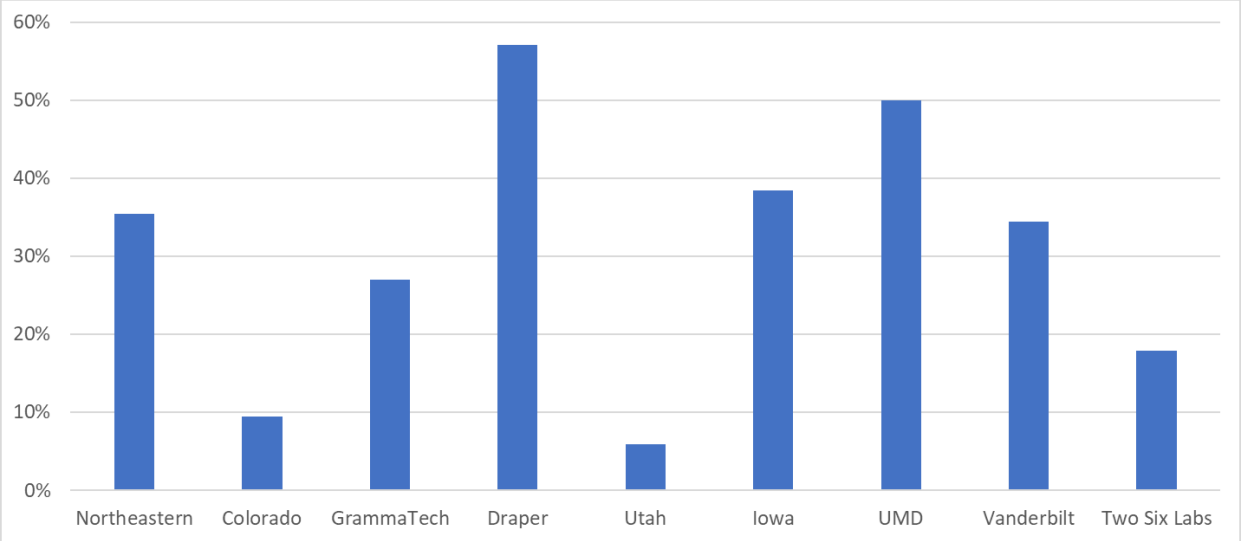


Figure 57: E5 Percentage-point difference between take-home and live-collaborative accuracies

4.5.2.1.3 Collaborative Impact Analysis

The qualitative and quantitative impact analysis performed in this section show that Colorado and Vanderbilt combined to have the most impact on the collaborative engagement, with UMD, Draper, and Two Six Labs having the least impact on the collaborative engagement.

To analyze the impact of different teams on the collaborative engagement, we initial used a set of metrics based on the performance of a Blue Team's collaborators during the live-collaborative engagement. Performance is measured based on a score, +1 for each correct response, -1 for each incorrect. The first metric looks at the total score change of all collaborators for a given team from the take-home to the live-collaborative engagement. With this first metric a team can get a question incorrect and yet benefit from fellow collaborators getting this question correct or a team can get a question correct and yet be penalized for fellow collaborators getting this incorrect. This metric is based on the expectation that an impactful team will influence fellow collaborators to get a question correct (larger score change) and a less impactful team will either fail to convince fellow collaborators or convince them to answer incorrectly.

A second metric looks at the total score change, but only for the collaborators with the same live-collaborative response as a given team. This ensures that a team only benefits from getting the correct final response and convincing others of that correct response.

The third metric looks at the total score change, for collaborators with the same take-home response as a given team. The difficulty with this metric is that the first two metrics are based on the live-collaborative responses and this metric is based on the take-home responses. These two datasets are not directly comparable as these metrics are driven by the underlying accuracy in the engagement component and the accuracy on the live-collaborative component (90%) is far higher than the accuracy for the take-home component (57%). To allow for direct comparison, the resulting impact scores are normalized by the accuracy of the relevant engagement components. The results are shown below.

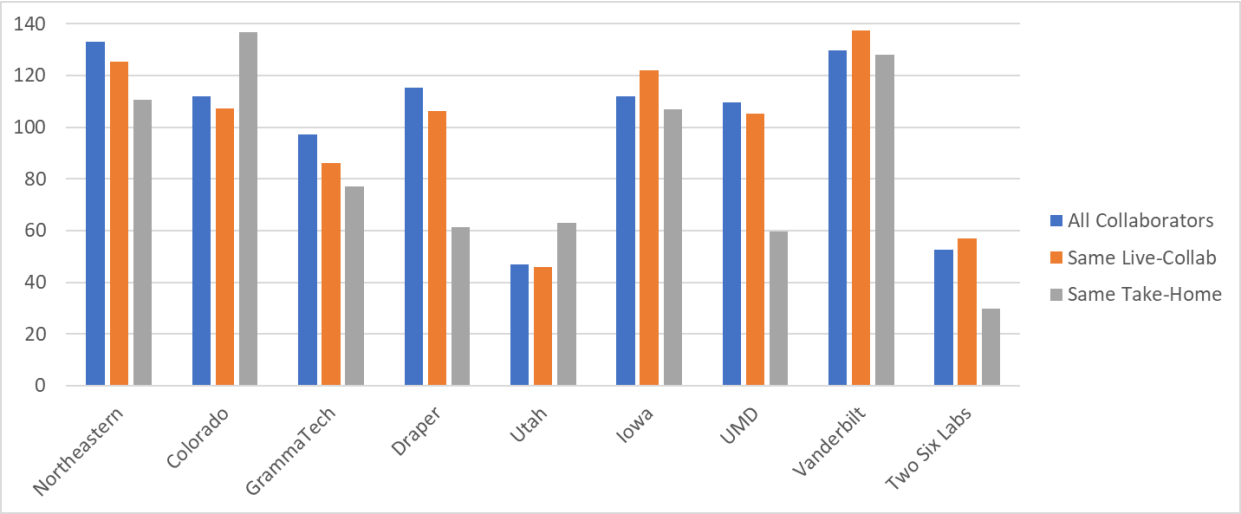


Figure 58: Normalized collaborative impact score based on three metrics

The first metric of all collaborators serves as the baseline measurement of a team's collaborators. The second metric of only collaborators with the same live-collaborative response indicates the performance of collaborators that sided with the team during the collaborative engagement. The relative difference between these two metrics indicates the performance of those that agreed with a team relative to the group of all collaborators. In this comparison, Iowa and Vanderbilt had the largest impact with their same-live-response score noticeably greater than their all-collaborators score. By contrast, Draper and GrammaTech had the smallest impact with their same-live-response scores noticeably less than their all-collaborators score.

A second impact comparison looks at the all collaborators score relative to the same-take-home (collaborators with the live collaborative response as a team's take-home response) score. With this measure, Colorado understandably has the largest impact based on their 74% take-home accuracy coupled with the large number of questions answered (23 questions, 74% of total questions). By contrast, Draper and UMD had the lowest impact which makes sense given their take-home accuracies of 24% and 36% respectively.

These impact metrics provide an initial idea of the impact of different teams on the collaborative engagement. However, these metrics don't truly capture how much a team's tools and thought process impacted fellow collaborators. To better capture the true impact, we looked through the take-home and live-collaborative responses and identified cases where a team's take-home responses are reflected in fellow collaborators responses. In this qualitative analysis the presence of a team's take-home response in another's live engagement response is noted as an influenced team. For each team a score of +1 is assigned each time an influenced team on a given question is evaluated as correct; conversely a score of -1 is assigned each time an influence team on a given question is evaluated as incorrect. The results of this more qualitative impact analysis are shown in Figure 59 below.

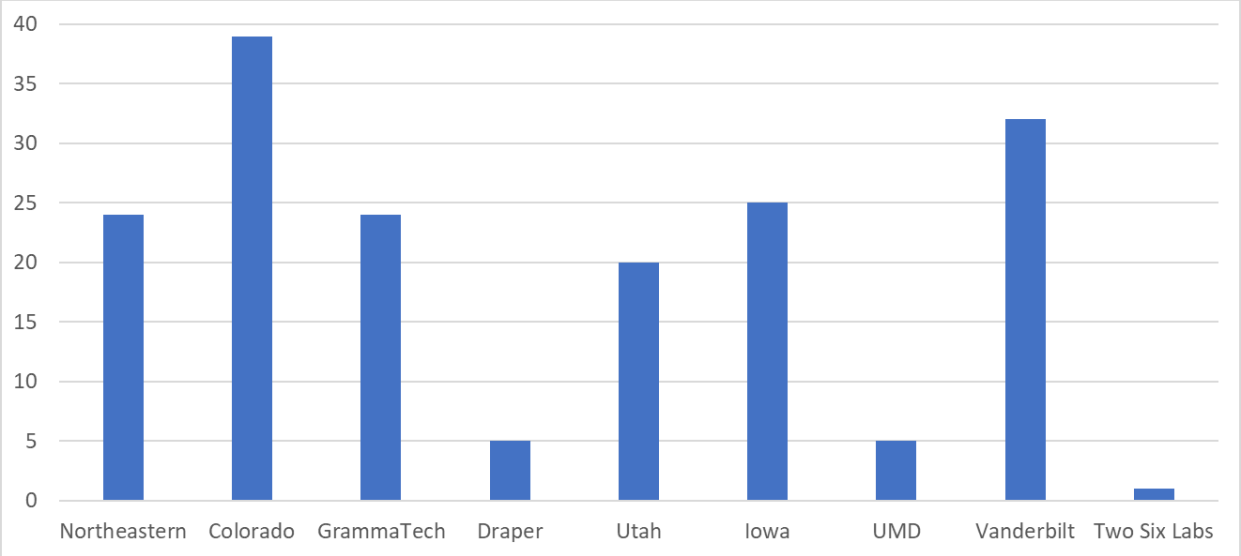


Figure 59: Qualitative live-collaborative impact scores (All responses)

This analysis shows that Colorado's take-home engagement responses had the largest positive impact on fellow collaborators. Vanderbilt is next, with Iowa 3rd. Utah's large influence despite only answering 7 questions is largely due to being the only team to identify the AC Space vulnerability in accounting wizard and causing all other teams to respond correctly during the collaborative engagement. UMD and Utah tied for the second lowest influence with Two Six Labs understandably last due to only being allowed to collaborate on "No" responses and responses with no exploit.

One of the main goals of STAC is to develop tools that not only identify vulnerabilities but can rule out vulnerabilities given certain bounds. Additionally, convincing others of a "No" response is more difficult than convincing others of a "Yes" response, as in the case of a "No" response, there is no exploit. The qualitative impact score data was therefore segmented to include only cases where a "No" response was provided. The results (Figure 60) show that Vanderbilt had the largest impact in convincing others of "No" responses, followed by Colorado and GrammaTech. Draper, UMD, and Utah had no impact in this segmentation of qualitative analysis.

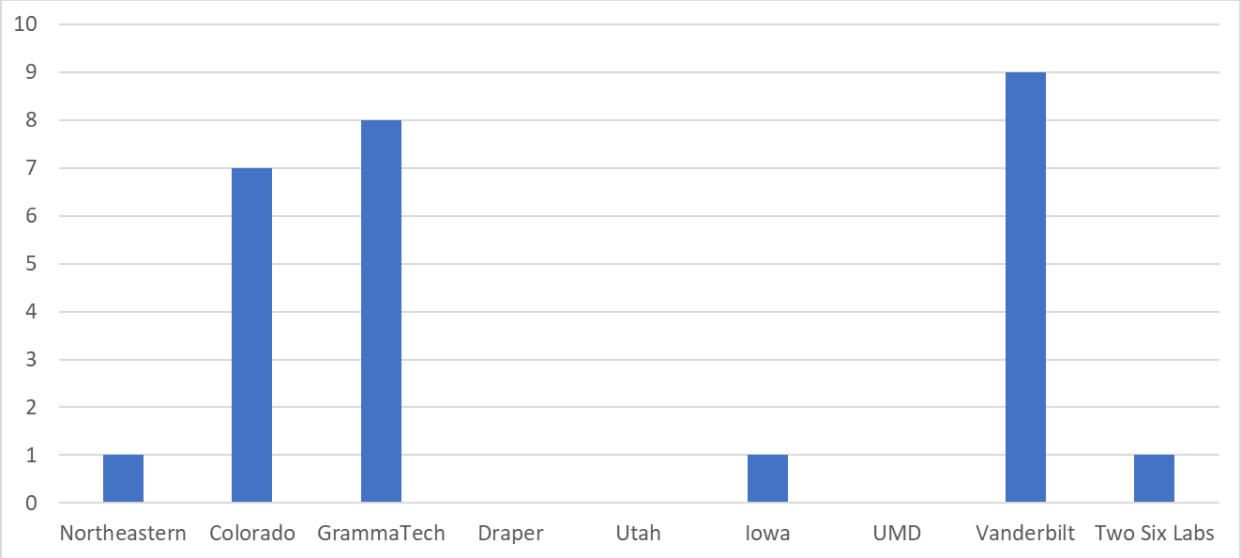


Figure 60: Qualitative live-collaborative impact scores ("No" responses)

4.5.2.1.4 Blue Team Responses

The following tables show the Engagement 5 responses for each team. Correct answers are in green and incorrect ones are in red, and unanswered questions are in yellow.

Program	Question	Type	Vulnerable	Northeastern	Colorado	GrammaTech	Draper	Utah	Iowa	UMD	Vanderbilt	Two Six Labs
accounting_wizard	Question_011	SC Time	Y	Y		N	Y		Y	Y	N	Y
accounting_wizard	Question_014	AC Space	Y	N	N	N	N	Y	N	N	N	N
accounting_wizard	Question_023	AC Time	Y	N	Y	Y	N			Y	Y	Y
accounting_wizard	Question_026	SC Space	Y	Y	Y	Y	N	Y	Y	N	Y	Y
battleboats_1	Question_006	AC Time	Y	Y	Y		N	Y	N	Y	N	Y
battleboats_1	Question_028	SC Space	N	N			N		Y		N	N
battleboats_2	Question_010	SC Time	Y	Y	Y	Y	N		Y		Y	Y
battleboats_2	Question_021	AC Space	Y	Y	Y		Y		N	Y	N	Y
braidit_1	Question_002	SC Time	N	N	Y				N		N	N
braidit_1	Question_007	SC Space	Y	Y	Y	Y			Y		Y	Y
braidit_1	Question_017	AC Space	Y	N	Y				Y		Y	Y
braidit_2	Question_001	SC Space	Y	Y	Y	Y			N		Y	Y
braidit_2	Question_022	AC Time	Y	N	Y				Y		N	Y
ibasys	Question_015	SC Space	Y	N		N	N		N	N	Y	Y
ibasys	Question_024	SC Time	N	N		N	Y		N	Y	N	N
ibasys	Question_029	AC Space	Y	Y	Y	N	Y		N	N	Y	N
medpedia	Question_009	SC Space	Y	Y	N	N	N		Y	N	Y	N
poker	Question_004	SC Space	N	Y	N	N	N		Y	N	Y	N
poker	Question_031	SC Time	Y	N	N	N	Y			N	N	Y
searchable_blog	Question_018	AC Space	N	N		N		N	N	N		N
searchable_blog	Question_027	AC Time	Y	Y	Y	N	N	Y	Y	Y	Y	Y
simplevote_1	Question_019	AC Time	Y	N	Y		N		Y	N	Y	
simplevote_1	Question_025	SC Time/Space	N	N	Y		Y				N	
simplevote_2	Question_005	SC Space	Y	N					Y	Y	N	
simplevote_2	Question_008	AC Space	Y	N	Y							Y
simplevote_2	Question_016	SC Space	Y	N					N	N	Y	Y
stacsql	Question_013	AC Space	Y	N	Y	Y	N	Y		Y	Y	N
stegosaurus	Question_003	SC Time	N	Y	N	N	Y	Y	Y	Y	N	Y
stufftracker	Question_030	AC Space	Y	Y	Y	Y	Y		Y	N	N	Y
tawa_fs	Question_012	AC Time	Y	N	Y	N	Y		Y	N	N	Y
tawa_fs	Question_020	AC Space	Y	Y		Y			Y	N	N	N

Figure 61: E5 Take-home responses (red – incorrect, green – correct, yellow – no response)

Figure 62 below shows the Blue Teams responses updated for the live-collaborative engagement.

Program	Question	Type	Vulnerable	Northeastern	Colorado	GrammaTech	Draper	Utah	Iowa	UMD	Vanderbilt	Two Six Labs
accounting_wizard	Question_011	SC Time	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
accounting_wizard	Question_014	AC Space	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
accounting_wizard	Question_023	AC Time	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
accounting_wizard	Question_026	SC Space	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
battleboats_1	Question_006	AC Time	Y	Y	Y		Y	Y	Y	Y	Y	Y
battleboats_1	Question_028	SC Space	N	N			N		N		N	N
battleboats_2	Question_010	SC Time	Y	Y	Y	Y	Y	Y	Y		Y	Y
battleboats_2	Question_021	AC Space	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
braidit_1	Question_002	SC Time	N	N	Y				N		N	N
braidit_1	Question_007	SC Space	Y	Y	Y	Y			Y		Y	Y
braidit_1	Question_017	AC Space	Y	Y	Y	Y			Y		Y	Y
braidit_2	Question_001	SC Space	Y	Y	Y	Y			Y		Y	Y
braidit_2	Question_022	AC Time	Y	Y	Y	Y			Y		Y	Y
ibasys	Question_015	SC Space	Y	N		N	N		Y	N	Y	Y
ibasys	Question_024	SC Time	N	N		N	Y		N	N	N	N
ibasys	Question_029	AC Space	Y	Y	Y	Y	Y		Y	Y	Y	Y
medpedia	Question_009	SC Space	Y	Y	Y	N	Y		Y	Y	Y	Y
poker	Question_004	SC Space	N	Y	N	N	N		N	N	N	N
poker	Question_031	SC Time	Y	N	N	N	Y			Y	N	Y
searchable_blog	Question_018	AC Space	N	N		N		N	N	N		N
searchable_blog	Question_027	AC Time	Y	Y	Y	Y	Y	Y	Y	Y	Y	Y
simplevote_1	Question_019	AC Time	Y	Y	Y		N		Y	Y	Y	
simplevote_1	Question_025	SC Time/Space	N	N	N		N				N	
simplevote_2	Question_005	SC Space	Y	Y					Y	Y	Y	
simplevote_2	Question_008	AC Space	Y	Y	Y							Y
simplevote_2	Question_016	SC Space	Y	Y					Y	N	Y	Y
stacsql	Question_013	AC Space	Y	Y	Y	Y	Y	Y		Y	Y	Y
stegosaurus	Question_003	SC Time	N	N	N	N	N	N	N	Y	N	Y
stufftracker	Question_030	AC Space	Y	Y	Y	Y	Y		Y	Y	Y	Y
tawa_fs	Question_012	AC Time	Y	Y	Y	Y	Y		Y	Y	Y	Y
tawa_fs	Question_020	AC Space	Y	Y		Y		Y	Y	Y	Y	Y

Figure 62: E5 Live-collaborative responses (red – incorrect, green – correct, yellow – no response)

4.5.2.1.5 Categories of Incorrect Responses

Incorrect responses by Blue Teams fall into four categories. These categories are not identical for AC and SC questions, but they mirror each other. For AC questions, the incorrect Blue Team response categories are:

AC 5. Did not follow directions or violated the question definitions
AC 6. Failed to identify the complexity within the challenge program
AC 7. Failed to understand the input to the complexity control flow of the challenge program
AC 8. Failed to properly determine the complexity bounds of an identified complexity

For SC questions, the incorrect Blue Team response categories are:

SC 5. Did not follow directions or violated the question definitions
SC 6. Failed to identify the secret or the secret location within the challenge program
SC 7. Failed to identify the side channel (processing conditioned on the secret value)
SC 8. Failed to properly analyze the side channel strength

An incorrect response can fit into multiple categories e.g. given strong side channel a team identifies a weaker side channel. This is classified as both a failure to identify the intended side channel vulnerability and a failure to correctly determine strength.

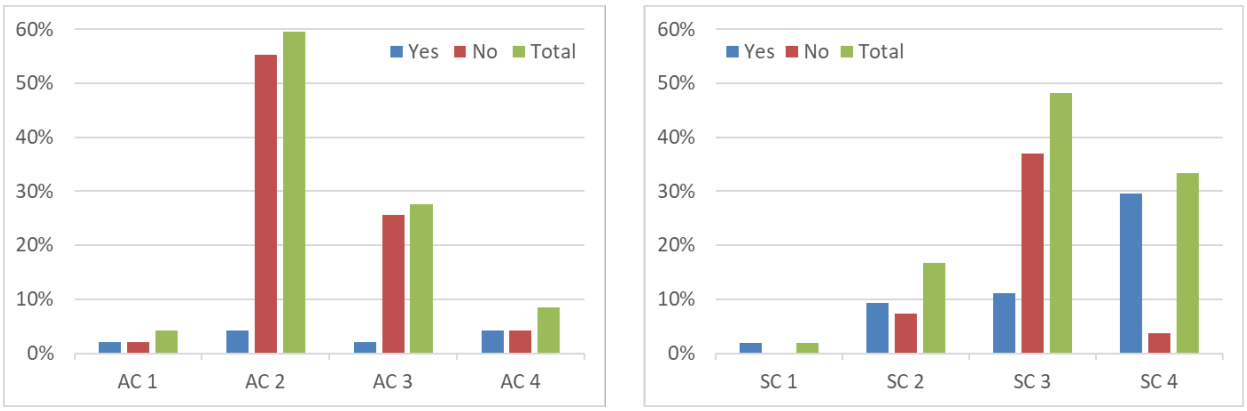


Figure 63: E5 Take-home categories of incorrect responses; algorithmic complexity (left); side channel (right)

The data shows algorithm complexity questions were answered incorrectly primarily due to a failure to identify the complexity followed by a failure to understand the input to the complexity control flow. This resulted in teams incorrectly declaring the challenge programs non-vulnerable.

In analyzing side channel vulnerabilities during the take-home component of the engagement, teams had the most difficulty identifying the side channel. This overwhelmingly resulted in teams incorrectly declaring these challenge programs non-vulnerable. Blue Teams continue to have difficulty evaluating strength; when teams incorrectly evaluate strength, they incorrectly declare challenge programs as vulnerable to the identified weak side channel.

Engagement 5 contained more difficult to identify vulnerabilities, including vulnerabilities due to multi-threaded and non-persistent vulnerabilities. These coupled with evaluating non-local vulnerabilities and input guards present problems for Blue Teams.

4.5.2.2 AC Teams

The AC Teams (BBN and CyberPoint) produced 15 challenge programs for Engagement 5; resulting in 31 challenge questions. During the engagement, two unintended *out-of-scope* vulnerabilities were discovered in BBN challenge programs. In one of the CyberPoint challenge programs, BraidIt 1, an error invalidated three SC challenge questions and resulted in the re-issue of a repaired engagement article to Blue Teams. Both AC Teams had challenge program with an intended vulnerability, but the analysis we performed in addition to the analysis performed by the AC Teams failed to identify that the operational budget was insufficient to determine the worst-case secret.

The figures below illustrate the trends in the AC Teams' performances on STAC engagements.

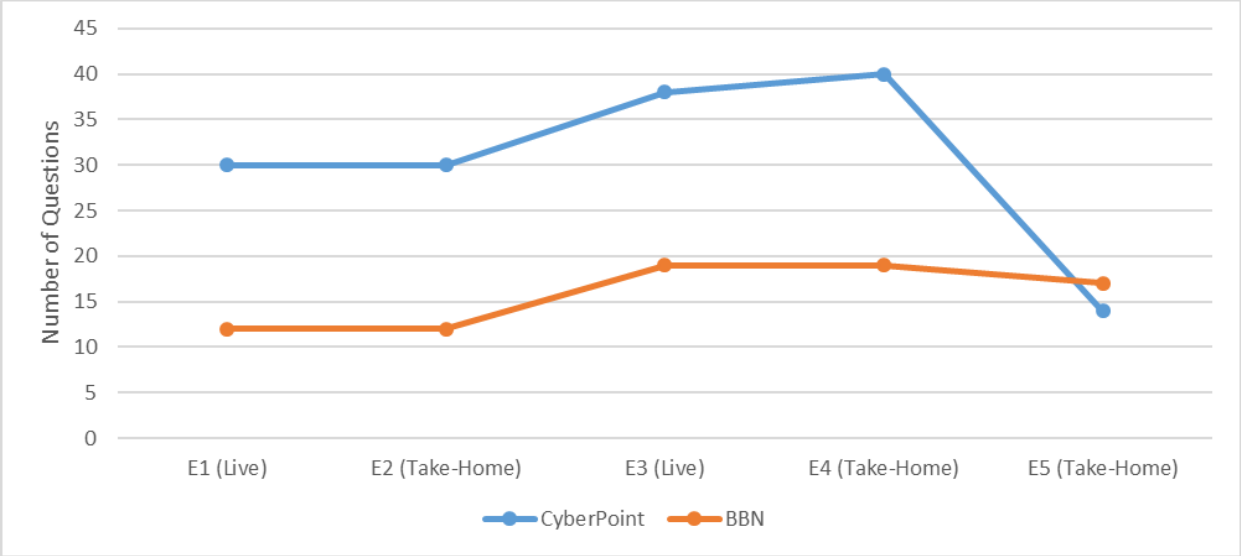


Figure 64: Number of engagement challenge questions posed per engagement

In Engagements 1 through 4, CyberPoint provided more challenge questions than BBN. CyberPoint's use of multiple variants of the same underlying challenge programs allowed for more questions to be posed. By contrast, each of BBN's challenges was a unique application. Engagement 5 was the first engagement with more BBN than CyberPoint questions.

In Engagements 1 through 4, CyberPoint had more total question responses than BBN. In Engagement 5 this trend reversed. Between Engagements 1 and 4, the trend of responses to CyberPoint questions oscillates between the take-home and live engagements. The number of responses to CyberPoint and BBN questions is closest in the live engagements. As the BBN challenge programs are contain fewer lines of code, this is understandable. During the take-home engagements, with more time and more CyberPoint questions than BBN questions available, Blue Teams provide more responses to CyberPoint questions than to BBN questions.

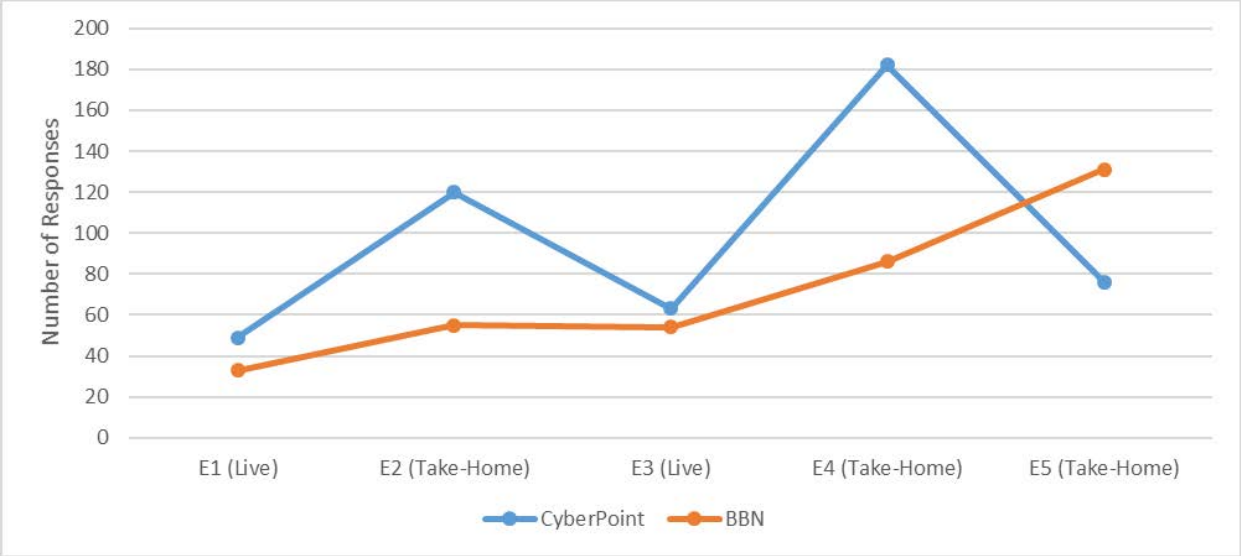


Figure 65: Number of Blue Team responses

Total number of question responses doesn't fully capture Blue Team affinity for one AC Team's challenge programs over another's. To allow for a more direct comparison, Figure 66 shows the percentage of available challenge questions for an AC Team answered by the Blue Teams. This metric better demonstrates Blue Team affinity and shows that, except for Engagement 4, Blue Teams answered a greater percentage of the available BBN questions than the available CyberPoint questions. In Engagement 4 both AC Teams were approximately even.

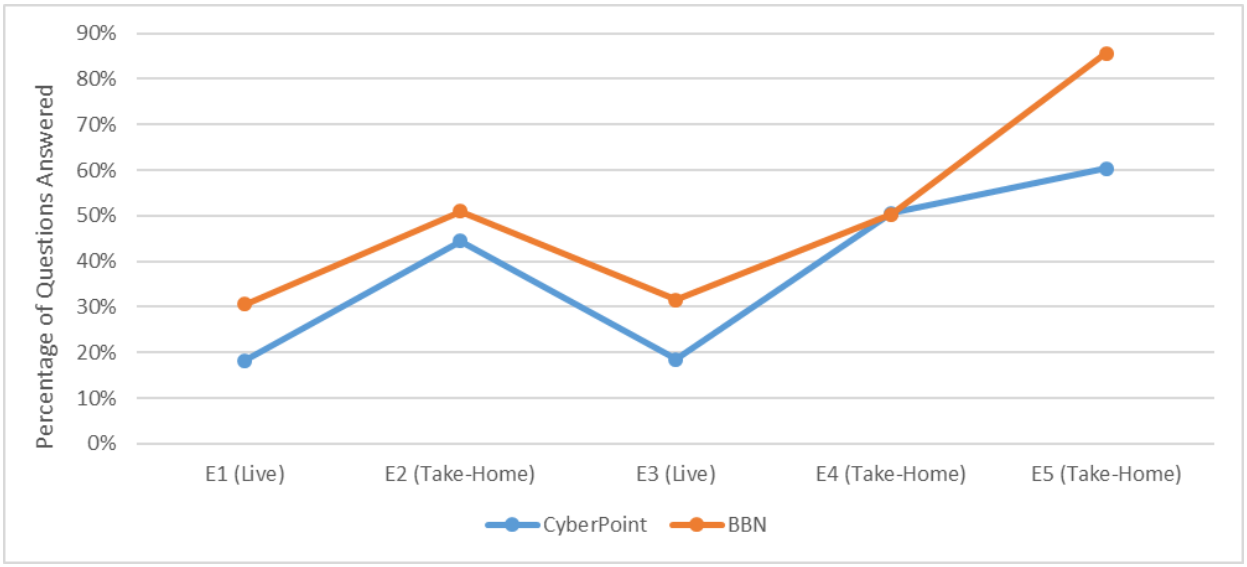


Figure 66: Percentage of available questions answered by Blue Teams

Figure 67 shows the accuracy of Blue Teams in answering each AC Team's challenge questions. Except for Engagement 4, Blue Teams have performed better on CyberPoint questions than on BBN questions. In Engagement 5, teams had an 8-percentage point better average accuracy on CyberPoint questions than on BBN questions.

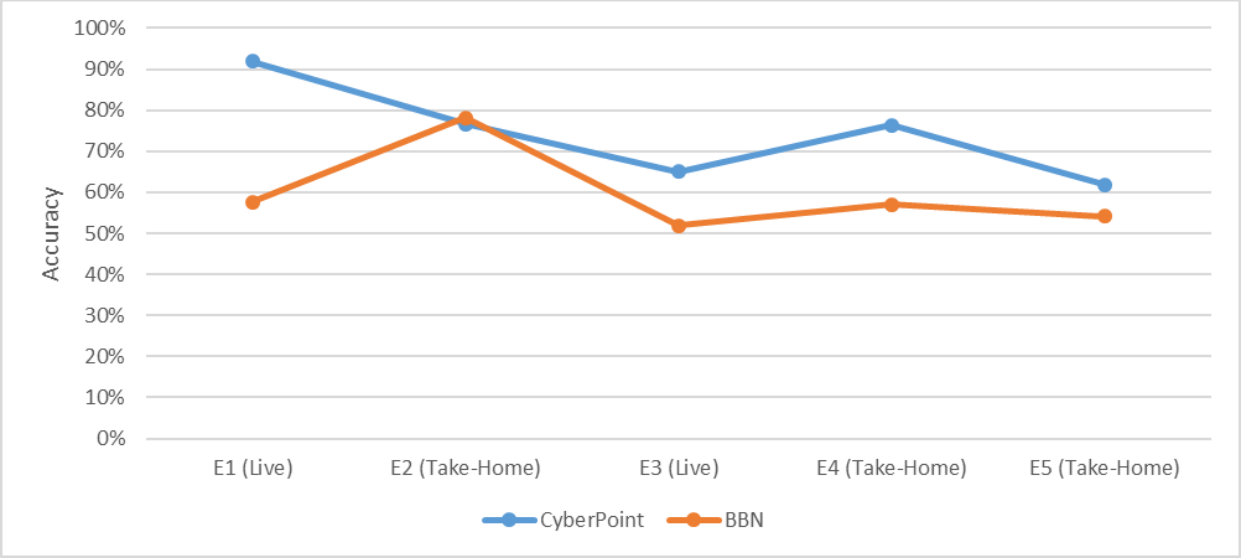


Figure 67: Blue Team accuracy on each AC Team's challenge programs

4.5.2.2.1 CyberPoint

CyberPoint provided 6 challenge programs for Engagement 5. Three CyberPoint challenge programs contained issues with one of the issues resulting in a re-issuing of the engagement article (repository containing the engagement challenge programs).

4.5.2.2.1.1 BraidIt Bug

Both variants of the BraidIt challenge program contained a bug that impacted three SC challenge questions. BraidIt is a peer-to-peer application that allows two users to play a game. A game consists of three rounds and in each round, a specified number of braids are randomly generated. One player selects and modifies one of the braids and the other player tries to determine the selected and modified braid. Both variants of BraidIt contained an intended SC Space vulnerability that allows a user to determine the braid selected and modified by an opponent. In the original application delivery, the attacker generates the braids for the benign user to select and modify. This allows the attacker to guarantee that the generated braids are not random and therefore trivially determine the secret. A late patch was made to the application such that the benign user's application generates the required random braids. The bug is that the java object for the selected braid is not copied prior to modification. As a result, when the braid is modified the original braid is lost. As the three challenge questions ask about the original braid selected for modification, the applications were trivially vulnerable or trivially non-vulnerable depending on the interpretation of the question. This bug was corrected with a one-line code change, but the engagement article had to be changed during the engagement.

4.5.2.2.1.2 SimpleVote Unintended Non-Vulnerable

The SimpleVote challenge program is a peer-to-peer and web-based electronic voting program. One of the intended vulnerable side channel questions asks if a target user's ballot can be viewed by an attacker. The target user's ballot can be obtained with their registration key (a single 12-character string with two components: $kstr$ – characters 1 to 6; $nstr$ – characters 7 to 12). The intended vulnerability leaks the $nstr$ component and with a one-to-one mapping between the $nstr$ and $kstr$ components the $kstr$ value can be determined. Armed with both the $nstr$ and $kstr$ values, the attacker can construct the registration key and view the target user's ballot. However, during a pre-processing step, a many-to-one mapping between the $kstr$ and the $nstr$ weakens the side channel. Vanderbilt discovered this issue during the take-home engagement and the Engagement 5 answer key was changed to reflect this.

Table 29: CyberPoint take-home responses

Challenge Program	Vulnerable Questions	Non-Vuln. Questions	Total Questions	Responses	Correct Responses	Accuracy
BattleBoats 1	1	1	2	13	9	69%
BattleBoats 2	2	0	2	14	9	65%
BraidIt 1	2	1	3	16	13	81%
BraidIt 2	2	0	2	11	5	45%
SimpleVote 1	1	1	2	10	5	50%
SimpleVote 2	3	0	3	12	6	50%
Total	11	3	14	76	47	62%

Table 30: CyberPoint live-collaborative responses

Challenge Program	Vulnerable Questions	Non-Vuln. Questions	Total Questions	Responses	Correct Responses	Accuracy
BattleBoats 1	1	1	2	13	13	100%
BattleBoats 2	2	0	2	17	16	94%
BraidIt 1	2	1	3	17	15	88%
BraidIt 2	2	0	2	12	9	75%
SimpleVote 1	1	1	2	10	9	90%
SimpleVote 2	3	0	3	12	11	92%
Total	11	3	14	81	73	90%

4.5.2.2.2 Raytheon/BBN Technologies

BBN provided 9 challenge programs for Engagement 5.

4.5.2.2.2.1 Unintended Vulnerabilities

Two BBN challenge programs (Stegosaurus and IBASys) contained unintended vulnerabilities that directly leaked the secret without a side channel. As the associated challenge questions asked about side channels, these vulnerabilities were evaluated as *out-of-scope*. Additionally, the Stegosaurus challenge program contained an intended vulnerability expected to be invariant to the size of the image. However, due to the implementation, for certain combinations of secret values

the side channel is of insufficient strength. Finally, the Accounting Wizard challenge program contained an unintended side channel vulnerability discovered by Iowa and Northeastern during the take-home engagement and verified by all teams except Colorado and Utah during the live-collaborative engagement.

4.5.2.2.2.2 Accounting Wizard Unintended Vulnerable

The Accounting Wizard application is an application with three user-levels. Employees are the least privileged users with the ability to charge items and hours only on authorized projects. Employees don't have access to the total expenditure limit (Project Expenditure Limit); however, an unintended vulnerability allows an attacker with employee privileges to leak this information. In the case where no hours have been charged by an employee on a project, an attempt to add hours causes an exception to be thrown. In handling the exception, the newly submitted hours are directly compared to the secret Project Expenditure Limit in FileStore.hoursFit() method. This comparison triggers writes to the log file if the newly submitted hours are less than or equal to the secret Project Expenditure Limit. The number of log writes that occurs controls the strength of the side channel. As delivered, the number of log writes is insufficient to distinguish when newly submitted hours exceed versus don't exceed the secret Project Expenditure Limit. However, the side channel strength can be amplified by either using the operational budget to expense a large number of items, increasing the number of log writes, or by using an intended AC Space vulnerability to trigger verbose logging an expensing approximately 50 items.

Table 31: BBN take-home responses

Challenge Program	Vulnerable Questions	Non-Vuln. Questions	Total Questions	Responses	Correct Responses	Accuracy
Accounting Wizard	3	1	4	32	15	47%
IBASys	2	1	3	22	11	50%
Medpedia	1	0	1	8	3	38%
Poker	1	1	2	15	7	47%
Searchable Blog	1	1	2	15	13	87%
STAC SQL	1	0	1	8	5	63%
Stegosaurus	1	0	1	9	5	56%
StuffTracker	1	0	1	8	5	63%
Tawa FS	2	0	2	14	7	50%
Total	13	4	17	131	71	54%

Table 32: BBN live-collaborative responses

Challenge Program	Vulnerable Questions	Non-Vuln. Questions	Total Questions	Responses	Correct Responses	Accuracy
Accounting Wizard	3	1	4	36	33	92%
IBASys	2	1	3	22	17	77%
Medpedia	1	0	1	8	7	88%
Poker	1	1	2	15	10	67%
Searchable Blog	1	1	2	15	15	100%
STAC SQL	1	0	1	8	8	100%
Stegosaurus	1	0	1	9	9	100%
StuffTracker	1	0	1	8	7	88%
Tawa FS	2	0	2	15	15	100%
Total	13	4	17	136	121	89%

4.6 Engagement 6

4.6.1 Engagement Plan

4.6.1.1 Purpose and Program Scope

4.6.1.1.1 Program Scope

This Engagement Plan is for Engagement 6, a combined take-home and live collaborative engagement. Take-home engagement answers must be submitted by noon Eastern time on August 1st, 2018. The live collaborative engagement will take place from August 21st – August 23rd, 2018; answers must be submitted at the end of the live collaborative engagement on August 23rd, 2018. Answers should be sent to evan.fortunato@apogee-research.com and kwame.ampeh@apogee-research.com or uploaded to Apogee's Gitlab server.

The document is structured into the following sections.

Section 1: Purpose and Program Scope
Section 2: Operational Definitions and Research Hypotheses
Section 3: Engagement 3 Plans and Logistics
Section 4: Post-Engagement Analysis Plans

4.6.1.2 Operational Definitions and Program Hypotheses

4.6.1.2.1 Operational Definitions of STAC Vulnerabilities

To guide the Engagements, the STAC AC, EL, and government teams have come up with operational definitions of vulnerability to algorithmic complexity attack and to side channels. The AC performers will use these definitions to determine when the vulnerabilities they've inserted into their challenge programs are strong enough for use in engagements. The Control and R&D Teams will use these operational definitions to determine when they've found a vulnerability worth reporting – i.e., if it meets the definition of a STAC vulnerability. The operational definition document, currently *2017-08-22-opdef-v08.1*, specifies the definitions for STAC vulnerabilities along with guidance for Blue team responses to engagement challenges.

4.6.1.2.1.1 Unintended Vulnerabilities in Challenge Programs

Many Algorithmic Complexity and Side Channel vulnerabilities may exist in each challenge program beyond those intentionally inserted. For example, a normal authentication function is technically a side channel that leaks a secret. However, most of these vulnerabilities are "weak" – e.g., for Algorithmic Complexity, the additional complexity is a small percentage of the normal complexity, and for Side Channels, the number of operations necessary to resolve the secret is impractically large.

The operational definitions defined above are designed to minimize these unintended vulnerabilities in the challenge programs with the use of strong vulnerability thresholds (e.g., adversary operation budget is small for Side Channels). These thresholds will serve to discount most of these "unintended" vulnerabilities as not meeting the STAC definition. However, some unintended vulnerabilities will occur that do meet the definition. A reported vulnerability that satisfies the operational definition criterion for a given challenge program is a correct response, regardless if it is an intended or unintended vulnerability on behalf of the AC teams. During the

analysis period after each engagement, the EL team will work with the Blue teams to verify that reported unintended vulnerabilities do actually meet the operational definition so that full credit is given. Unintended vulnerabilities can fall into two categories: those “in-scope” as per the STAC Operational Definitions document and those “out-of-scope” as per the STAC Operational Definitions document. Full credit will be given for all verified unintended vulnerability, whether they are “in-scope” or “out-of-scope.” In the case of “in-scope” unintended vulnerabilities, the engagement answer key will be updated to reflect the identified unintended vulnerabilities.

4.6.1.2.2 Research Hypothesis and Segmentation of Challenge Problems

Throughout the STAC program, the EL team will have a set of research hypotheses pertaining to performance of the tools with respect to different types of STAC vulnerabilities and challenge problems.

Example hypotheses include:

16. Complexity vulnerabilities can be due to: (1) a change in the complexity of the algorithm, (2) a change in the coefficients of a fixed complexity, or (3) a flawed data guard that allows processing of input data that shouldn’t be allowed. How do the different R&D techniques perform with respect to each of these types of complexity vulnerabilities?
17. Side channels can be weakened by increasing the number of secret symbols that are mapped to identical observed symbols or by increasing the width of the parent distributions of the observed symbols (causing overlaps of the of the samples from different observed symbol distributions). How do the different R&D techniques perform with respect to each of these challenges?
18. Some non-vulnerable programs are locally balanced (e.g., all paths through a sub-section of the binary consume equivalent time and space resources) while others are non-locally balanced (e.g., resource consumption is asymmetric on a set of paths in multiple parts of the program resulting in entire execution traces that are balanced). How do the different R&D techniques perform with respect to the locality of balancing of resource consumption?

In support of these hypotheses, each challenge problem will be assessed and categorized with respect to how results on the challenge problem will support resolution of each of the hypotheses. This segmentation of the challenge problems will not be exposed during the engagements, but will be tracked for post-engagement analysis.

During the engagements, each challenge problem will come with questions such as:

21) Does the challenge problem contain a time-based side channel?
22) Does the challenge problem contain a space-based side channel?
23) Does the challenge problem contain a time-based complexity vulnerability?
24) Does the challenge problem contain a space-based complexity vulnerability?

Note, that the questions will be more specific version of these classes of questions and include additional context information and could include questions regarding combined time and space vulnerabilities.

4.6.1.3 Engagement 6 Details and Logistics

4.6.1.3.1.1 Engagement 6 Details

Engagement 6 is a combined take-home and live collaborative engagement. The take-home engagement will take place from 7/16/2018 – 8/1/2018, while the live collaborative engagement will take place 8/21/2018 – 8/23/2018. The collaborative challenge questions will be those that received a non-uniform response (the same response is not provided by all teams that responded) during the take home as determined by the EL. If there are not enough questions available, additional challenge questions meeting certain criteria will be made available for collaboration. The **collaborative group** for each collaborative challenge question will comprise the teams that submit a response for that question during the take-home engagement and teams that demonstrate work on a question (via the justification) without providing a response. Following a collaboration, the collaborative group is not required to reach a consensus on a response. Both R&D teams and the Control team will participate in the live collaborative engagement; however, the control team will only be allowed to collaborate on collaborative questions if they answered "No" (not vulnerable) to the question or if they answered "Yes" (vulnerable) but do not have an exploit.

The PI meeting (8/20/2018) will take place before the live collaborative engagement (8/21/2018 –8/23/2018). At the PI meeting, the list of **collaborative groups** for each challenge question will be provided to Blue teams. Instead of the usual PI meeting presentations, R&D teams should display their research capabilities to potential collaborators [other teams], highlighting the strengths and noting the weaknesses of their tools. To this end, each R&D team should:

1. Provide a quick review of what has changed since last PI meeting
2. A walkthrough of a challenge question you feel highlights the strength of your system
3. A walkthrough of a challenge question you feel highlights a deficiency of your system
4. Any additional information you want to share within your allotted time

Note: For (1), the PM is your audience, but for (2) – (4), your peers (fellow Blue Teams) are your audience. The control team should only present on questions that they answered "No" or questions where they answered "Yes" but do not have an exploit.

4.6.1.3.1.2 Engagement 6 Schedule

The schedule below provides an overview of the milestones for Engagement 6, including the engagement itself and the post engagement evaluation.

- Take-home engagement: 7/16/2018 – 8/1/2018.
- PI Meeting: 8/20/2018 at 4100 Fairfax Blvd Suite 750, Arlington, VA 22203.
- Live collaborative engagement: 8/21/2018 – 8/23/2018 at 4100 Fairfax Blvd Suite 750, Arlington, VA 22203.
- Release Engagement Plans (EL team): Initial release on 4/25/2018. Post on GitLab Public_EL_Information Project.
- AC Teams deliver completed versions of all engagement challenges for review to the EL: 5/3/2018 by noon eastern time.
- AC Teams deliver final versions of all Engagement 6 challenges to the EL: 5/31/2018 by noon eastern time.
- The EL team releases the take-home Engagement 6 challenges to Blue teams on 7/16/2018.

- Blue Teams submit take-home Engagement 6 responses to the EL: 8/1/2018 by noon eastern time.
- Quick-look engagement results (EL team): initial quick-look results presented to DARPA after the live collaborative engagement on 8/23/2018. Reviewed results subsequently posted to each team's repo on GitLab by 9/24/2018.
- Blue teams have until 10/24/2018 to request evaluation clarifications and changes.
- Final Engagement 6 analysis report delivered to DARPA: 11/23/2018.

4.6.1.3.1.3 Engagement 6 Logistics

The August 20th PI meeting and the Engagement 6 live collaborative engagement will take place at 4100 Fairfax Blvd Suite 750, Arlington, VA 22203 (down the street from DARPA). The live collaborative portion of Engagement 6 will occur from Tuesday, August 21st, 2018 through Thursday August 23rd, 2018. The detailed schedule is:

- Monday, August 20th, 2018:
  - 7:00 AM: EL team arrives for setup.
  - 7:30 AM – 8:00 AM: STAC teams arrive for PI meeting.
  - 8:00 AM – 5:30 PM: PI meeting.
  - 5:30 PM – 7:00 PM: Additional Setup time. No overnight processing.
- Tuesday, August 21st, 2018: Day 1:
  - 8:30 AM: Engagement starts for all teams.
  - 12:00 Noon: Everyone pauses engagement for lunch.
  - 1:00 PM: Engagement restarts for all teams.
  - 4:30 PM: Day 1 engagement ends. Teams are welcome to leave processes running overnight, but no work should be done by the teams on any part of the problem (honor system).
- Wednesday, August 22nd, 2018: Day 2:
  - 8:30 AM: Engagement starts for all teams.
  - 12:00 Noon: Everyone pauses engagement for lunch.
  - 1:00 PM: Engagement restarts for all teams.
  - 4:30 PM: Day 2 engagement ends. Teams are welcome to leave processes running overnight, but no work should be done by the teams on any part of the problem (honor system).
- Thursday, August 23rd, 2018, Day 3:
  - 8:30 AM: Engagement restarts for all teams.
  - 12:00 Noon: Day 3 engagement ends.

Each team will bring 3-5 people (if you want to bring more than 5 or fewer than 3, please request this to both DARPA and the EL team).

If teams would like to do remote processing, they are welcome to under the following restrictions and understandings:

- While it is the intent to provide external connectivity, we cannot guarantee high quality of service connectivity during the event (dependent on engagement location infrastructure). If connectivity is required for your approach to work, please have backup plans available.
- No one not at the engagement should work on, touch or be involved with the analysis

effort in any way during the engagement.

4.6.1.4 Post Engagement Analysis Plans

4.6.1.4.1.1 Metrics

The following set of metrics will be used to evaluate Blue team performance in Engagement 6:

- True Positive Rate (Number of correct "Yes" responses/ Total number of vulnerable challenge questions)
- True Negative Rate (Number of correct "No" responses/ Total number of non-vulnerable challenge questions).

Note, True Positive Rate is equivalent to a Probability of Detection Metric while the True Negative Rate is equal to 1 – False Alarm Rate. Using the True Positive and True Negative Rate gives insight into the relative strengths of each Blue team with respect to finding and ruling out vulnerabilities but doesn't give insight into the speed of the Blue teams.

To account for speed [and breadth of Blue team tool capability], two different segmentations of the data will be considered. In the first segmentation, only properly justified answers will be considered (in the numerator) and only answered challenge questions will be considered (in the denominator). This will allow us to see the performance of each approach on the set of problems for which they chose to answer. In the second segmentation, properly justified answers will be considered (in the numerator) and all challenge questions will be considered (in the denominator). This second segmentation shows the benefits of speed [and breadth] as Blue teams that only answer a small fraction of the questions will not be able to score well under the second segmentation.

To achieve the segmentation, each Blue team's answers will be evaluated against the intended vulnerabilities of the challenge programs. In cases where a Blue team believes that there is a vulnerability when none was intended, the EL team will re-evaluate the challenge application (using the justification provided by the team) to determine if an unintended vulnerability is present in the challenge application. If there is an unintended vulnerability, the EL will determine there is an in-scope unintended vulnerability. If there is an in-scope unintended vulnerability, then the challenge program status will be updated to act as though that unintended vulnerability was intended, and all Blue teams' answers will be rescored against this updated status. If there is an unintended out-of-scope vulnerability, then the status of the challenge program will not be updated (so other Blue teams' scores will not change) but the team that found the unintended out-of-scope vulnerability will be scored as if the challenge program in question is vulnerable. Note, this means that the denominators of the metrics for each Blue team will be different even under the second data segmentation.

Next, for each Blue team, the EL will evaluate the justifications against the justification guidance provided. If the justification is sufficient, then the answer will be considered a justified answer and will be included in the justified answer analysis. Independent of the validity of the justification, if a question is answered, then the challenge program will be included in the denominator under the first data segmentation.

The performance of each Blue team during the take-home and live collaborative portions of Engagement 6 will be compared to evaluate the impact of the collaborative engagement. For each Blue team, an impact metric will be calculated.

Finally, for the take-home portion of Engagement 6, the EL asks each team to track and report the total hours spent on each challenge question. This information will not be used to scale scores; instead, it will be used to gain a better understanding of which types of questions are easier and which are more difficult for each team. This information should be submitted along with the Engagement 6 responses either at the end of the response sheet or as a separate document. For some challenge programs, a team may spend time familiarizing with the challenge program before beginning work on the challenge question(s). This time should be reported separately from the time spent on the challenge question(s) if possible; if this is not possible, this time can be combined with the time spent on the challenge question(s) for that challenge program. Whatever option is selected should be consistently applied to all challenge programs and challenge questions.

4.6.1.4.1.2 Initial Assessments and Refinements Based on Blue Team Feedback

After the live collaborative engagement (8/23/2018), the EL team will provide an initial assessment to DARPA of the performance of each team against the program metrics using the intended vulnerabilities (i.e., those vulnerabilities that were intentionally added to the challenge programs). Following the release of individual assessments to each team on 9/24/2018, teams will have until 10/24/2018 to request evaluation clarifications and changes.

4.6.2 Engagement Results

4.6.2.1 Overview

STAC Engagement 6 took place from July 16th, 2018 to August 1st, 2018 and from August 21st, 2018 to August 23rd, 2018. Engagement 6 maintained the same short take-home and live collaborative engagement structure as Engagement 5. Engagements 5 and 6 were designed to counter the pattern (and by extension Blue Teams' expectations) of an even distribution of vulnerable and non-vulnerable challenge programs. Engagement 5 was significantly biased towards vulnerable challenges (77% vulnerable); by contrast Engagement 6 was significantly biased towards non-vulnerable challenges (33% vulnerable).

During the take-home engagement, R&D Teams were outperformed by the control team with the R&D Teams achieving an average accuracy 69% while the control team achieved an 89% accuracy (with 38 of 43 questions answered). This highest performing R&D Team, Iowa State, achieved a take-home accuracy of 76% (with 38 of 43 questions answered), 13 percentage points fewer than the control team. To prevent the control team from over-influencing the collaborative engagement, the team was excluded from the live-collaborative engagement. Following the live collaborative engagement, as with Engagement 5, the average R&D Team accuracy increased by approximately 30 percentage-points to 98%.

During Engagement 6, Blue Teams discovered unintended vulnerabilities that impacted 9 challenge questions (4 AC Space and 5 AC Time). Despite the control team identifying an out-of-scope unintended vulnerability to bypass the authentication on one of the challenge programs, no unintended side channel vulnerabilities were discovered.

In Engagement 6 (E6), 43 questions were asked about 15 challenge programs. Of the 15 challenge programs, 7 were provided by CyberPoint and 8 were provided by Raytheon/BBN Technologies (BBN). The breakdown of challenge questions by question type (Algorithmic Complexity (AC) in Time, AC in Space, Side Channel (SC) in Time, SC in Space, and SC in Space and Time) and vulnerability category (Vulnerable and Non-Vulnerable) is shown below.

Table 33: How many questions were asked for a given question type and intended question result

Question Type	Question Result	Number of Questions
Algorithmic Complexity in Time	Positive	5
Algorithmic Complexity in Space	Positive	2
Side Channel in Time	Positive	0
Side Channel in Space	Positive	1
Side Channel in Time and Space	Positive	0
Algorithmic Complexity in Time	Null	5
Algorithmic Complexity in Space	Null	6
Side Channel in Time	Null	12
Side Channel in Space	Null	7
Side Channel in Time and Space	Null	5

Eight R&D Teams and one control team participated in Engagements 1-5; however, in Engagement 6, six R&D Teams and one control team, collectively known as the Blue Teams, participated in the engagement. In total, the Blue Teams attempted 262 questions during the take-home engagement and the live-collaborative engagement (with 224 of those attempted by the R&D Teams). The average accuracy of the Blue Teams was 72% for the take-home component and 97% for the live component (69% and 98% for just the R&D Teams in the take-home and live components respectively). Note that due to the strong bias in the distribution of vulnerable versus non-vulnerable challenge questions, the accuracy data must be jointly considered with the True Positive Rate (TPR) and True Negative Rate (TNR) statistics discussed in Sections 1.1 and 1.2.

Table 34: The different teams that participated in Engagement 6

Blue Teams	Research Teams	University of Colorado Boulder (UC Boulder)
		GrammaTech
		Iowa State University (ISU)
		Northeastern University (NEU)
		University of Utah
		Vanderbilt University
	Control Team	Two Six Labs
Red Teams		CyberPoint
		BBN

A breakdown of the number of questions answered for each category and the resulting accuracy following a review by the EL is shown in Table 35.

Table 35: The distribution of Blue Team responses across different categories in the take-home component of E6

	Total Responses	Correct Reviewed Responses	Accuracy
SC Space	48	37	77%
SC Time	69	62	90%
SC Space/Time	26	25	96%
AC Space	54	28	49%
AC Time	65	36	51%

Table 36: The distribution of Blue Team responses across different categories in the live-collaborative component of E6

	Total Responses	Correct Reviewed Responses	Accuracy
SC Space	48	44	92%
SC Time	69	69	100%
SC Space/Time	26	26	100%
AC Space	54	52	96%
AC Time	65	63	97%

In the take-home component of Engagement 6, the Blue Teams had the highest accuracy for SC Time/Space questions with a 96% accuracy. The teams' performance across the other categories ranged from 49% to 90%. In contrast, following the live-collaborative component of the engagement the Blue teams' collective accuracy in all categories was greater than 96%.

4.6.2.1.1 Take-Home Engagement

This section evaluates Blue Team performance on a set of metrics measuring overall accuracy and performance on vulnerable versus non-vulnerable challenge questions, while accounting for

the number of responses provided. These metrics applied to the take-home component of Engagement 6 show that the control team outperformed the R&D Teams, with Iowa leading the performance of the R&D Teams. These metrics don't provide a complete picture of team performance; we must also consider a combination of these metrics and the quality of the responses provided by the teams coupled with an understanding of their research approaches. When these are considered GrammaTech, Iowa and Vanderbilt are in the first tier of research performers; Colorado is in the second tier; and Northeastern and Utah are in the third tier.

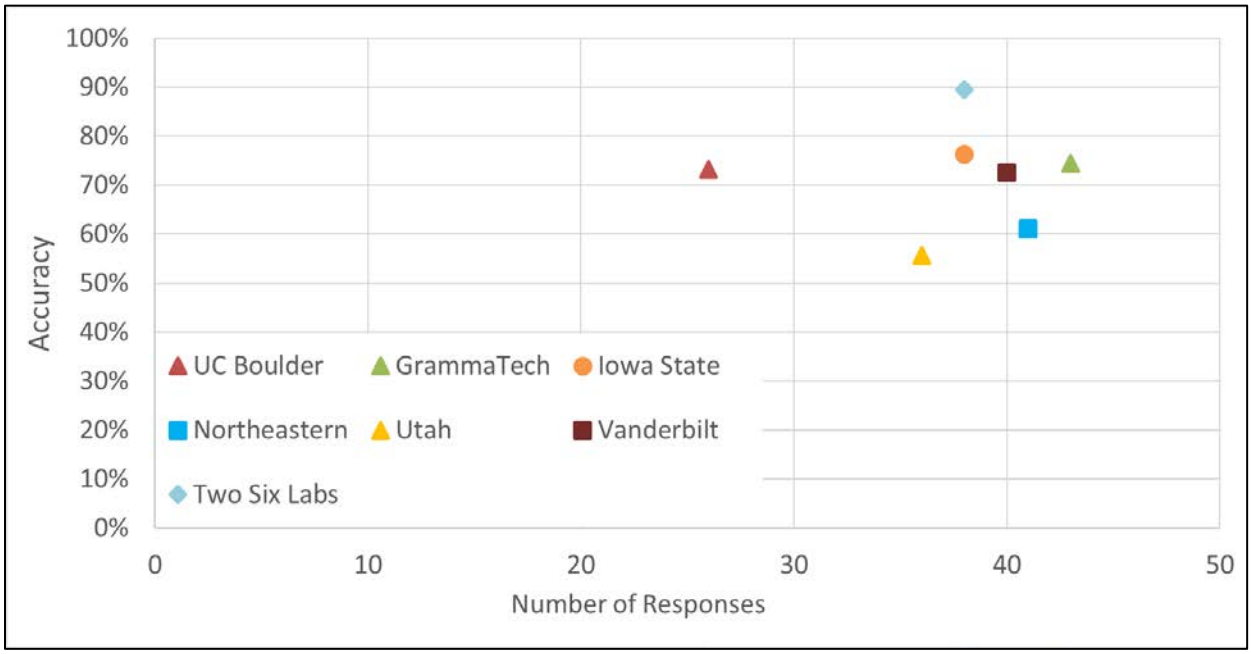


Figure 68: E6 take-home reviewed accuracy versus number of responses.

Engagement 6 contained a distribution of intended vulnerable and intended non-vulnerable challenge programs. During an engagement, Blue Teams may discover unintended vulnerabilities or discover that an intended vulnerability is not sufficiently strong. These discoveries are classified as *in-scope* if they satisfy the STAC operational definition requirements, and *out-of-scope* if they do not. *In-scope* discoveries result in changes to the engagement answer key for all teams. *Out-of-scope* discoveries only result in changes to the engagement answer key for the team that made the discovery. The engagement response review process accounts for the justifications in evaluating the binary (vulnerable/non-vulnerable) responses.

The plot of the take-home reviewed accuracy versus number of responses (Figure 68) shows that Two Six Labs had the highest accuracy of 89% while answering the 4th most questions (38 of 43). Iowa, GrammaTech, Vanderbilt, and Colorado had accuracies between 73% and 86%. Utah had the lowest accuracy of 56%.

STAC engagements test Blue Team's ability to identify vulnerabilities but place more focus on the ability to rule out vulnerabilities with sufficiently high accuracy. The metric of accuracy fails to capture how teams perform against vulnerable versus non-vulnerable challenge questions. As

with prior engagements, we computed the true positive rate (TPR) as a measure of a team's ability to identify vulnerabilities and the true negative rate (TNR) as a measure of a team's ability to rule out vulnerabilities. The TPR is the number of correct "Yes" responses divided by the number of vulnerable challenge questions. The TNR is the number of correct "No" responses divided by the number of non-vulnerable challenge questions.

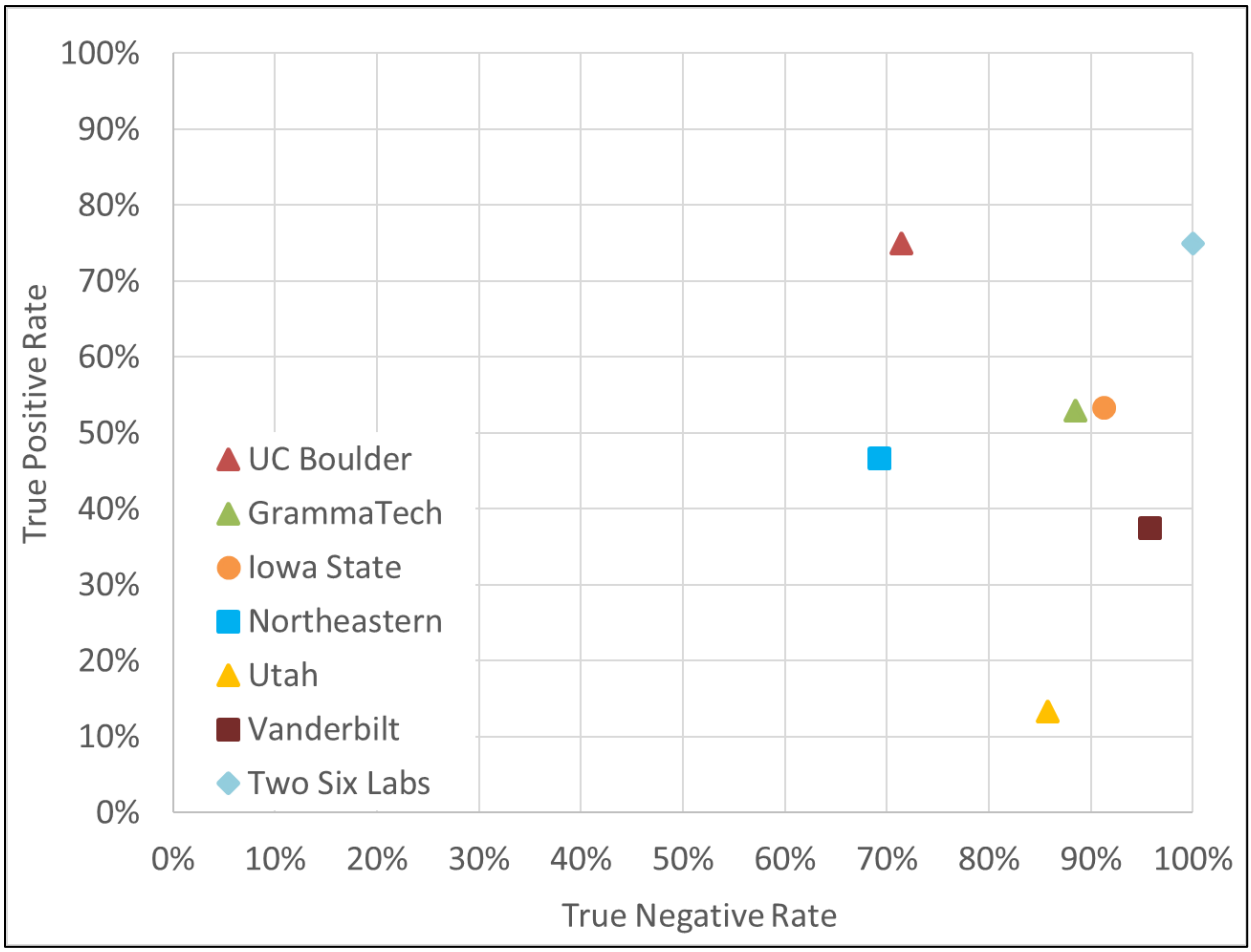


Figure 69: E6 take-home TPR vs TNR segmentation of answered questions

A ROC-curve style plot of TPR versus TNR is used to visualize the relative performances of the different Blue Teams. First, we look at Blue Team performance against the set of questions answered by each team. With the bias towards non-vulnerable challenge questions in this engagement, the TPR versus TNR plot distinguished the performance of the R&D Teams in identifying versus ruling out vulnerability – in particular, the clustered performances of Iowa, GrammaTech, Northeastern, and Vanderbilt.

The results shown in Figure 69 shows that Two Six Labs led all performers with the highest TNR of 100% and the tied highest TPR of 75%. Colorado and Vanderbilt are on opposing sides TPR and TNR performance. Colorado has the tied highest TPR of 75% but the second lowest TNR of 71% by contrast Vanderbilt leads the R&D Teams with the highest TNR of 96%, but the

second lowest TPR of 38%. This data corresponds to the relative strengths of their approaches, with Colorado focused on identifying vulnerabilities and Vanderbilt more capable of ruling them out. Iowa and GrammaTech have similar TPR and TNR performances – separated by 2.8 percentage-points in TNR and 0.4 percentage-points on TPR. Northeastern had the lowest TNR of all performers of 69% coupled with a 47% TPR. Utah had a 13% TPR and an 86% TNR.

The segmentation of answered questions does not account for the percentage of total questions answered by a Blue Team. To account for this, we use the same metrics of TPR and TNR but treat unanswered questions as incorrect. The result of this segmentation shown in Figure 70 has Two Six Labs leading all performers in TPR (71%) and GrammaTech leading all performers in TNR (89%). Colorado has the lowest TNR (36%) and Utah the lowest TPR (12.5%).

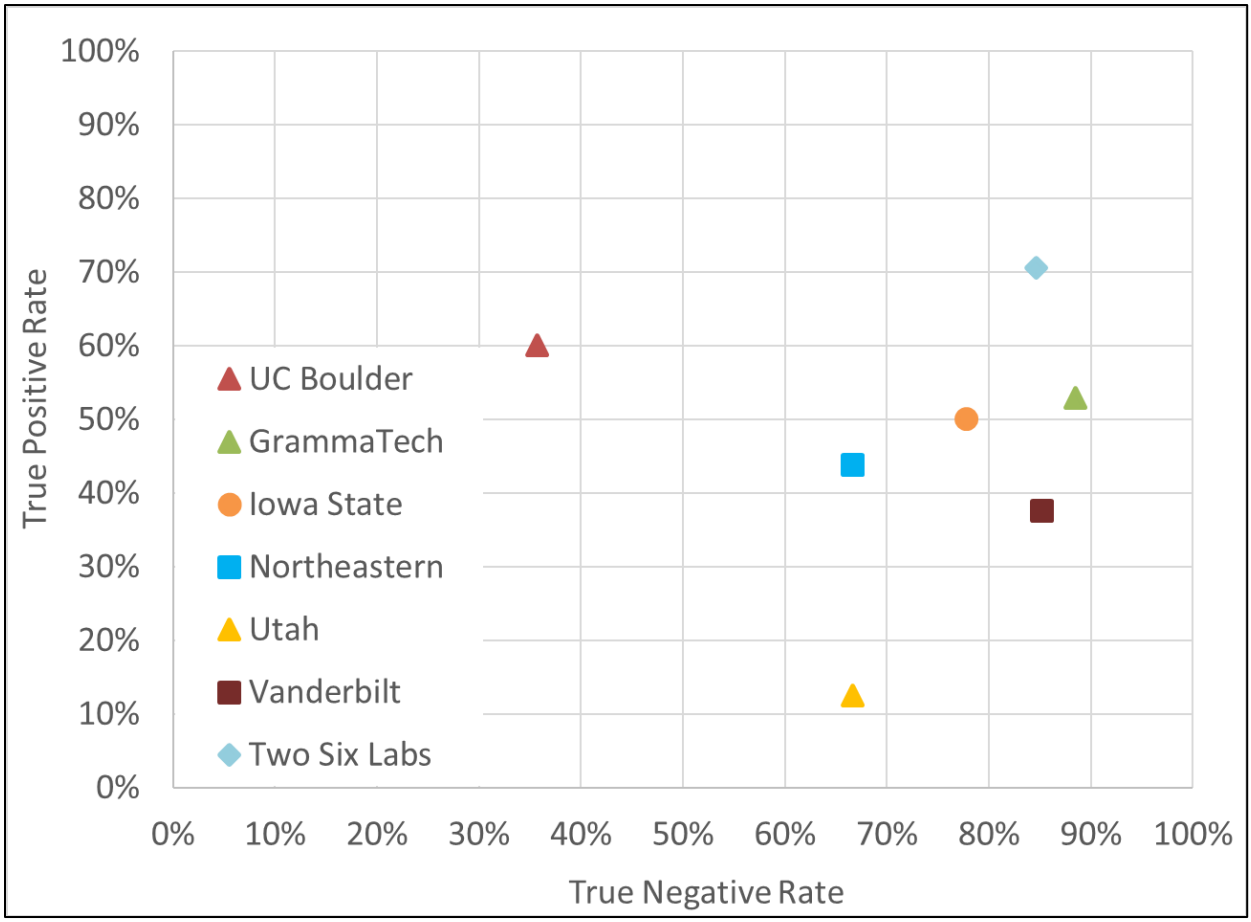


Figure 70: E6 take-home TPR vs TNR segmentation of all questions

4.6.2.1.2 Live-Collaborative Engagement

During the live-collaborative engagement, the six R&D Teams collaborated to improve their take-home performances and provide a measure of the collective STAC research program's performance. The figure below shows the total accuracies of the Blue Teams. As the control team did not participate in the collaborative engagement, their performance from the take-home to the live-collaborative engagement did not change.

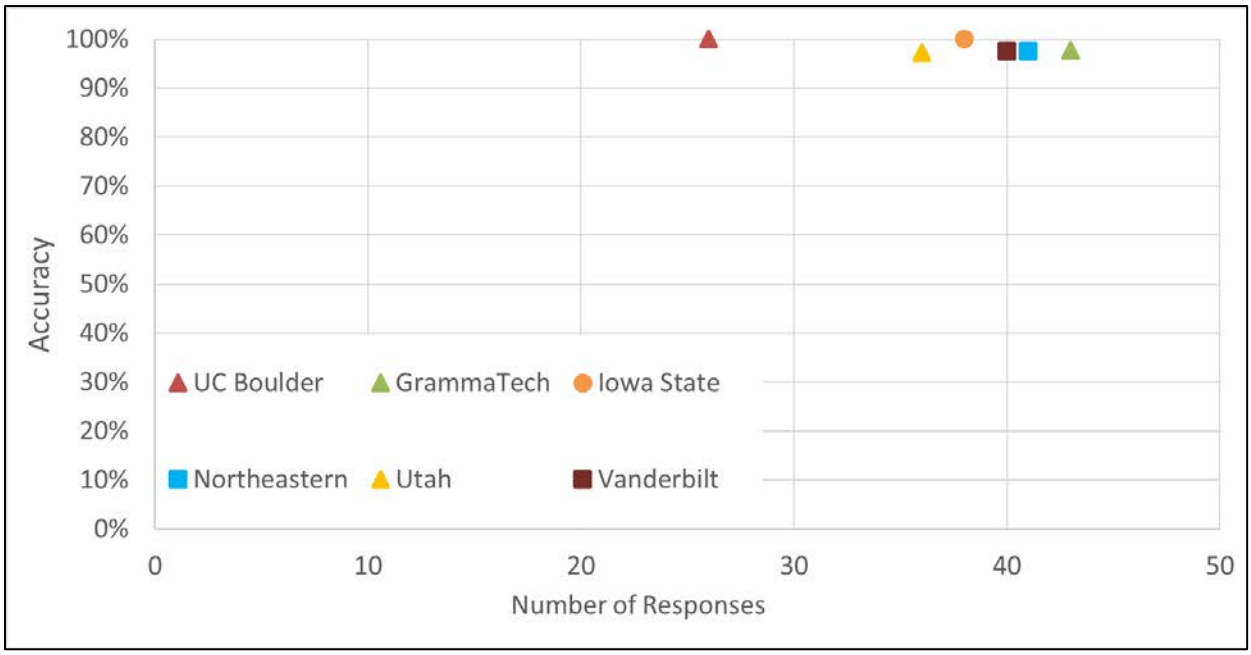


Figure 71: E6 live-collaborative reviewed accuracy versus number of responses.

As expected, the collaboration between the R&D Teams resulted in all the R&D Teams outperforming the take-home result of the control team. The combined R&D Team accuracy increased by 29 percentage-points from 69% in the take-home to 98% in the live-collaborative engagement. Iowa State and Colorado achieved a 100% accuracy after the collaborative engagement, with Utah having the lowest R&D Team collaborative accuracy of 97%. Vanderbilt, GrammaTech, and Northeastern tied, achieving a 98% collaborative accuracy. In the plot of TPR vs TNR for the set of answered questions, all the Blue Teams achieved a 100% TNR with the teams only separated by TPR. Iowa and Colorado achieved a 100% TPR, Vanderbilt and GrammaTech achieved a 94% TPR, and Utah a 93% TPR.

Against the segmentation of all questions, GrammaTech, Iowa, and Vanderbilt tied on TPR. Colorado was the lowest performer as they answered the fewest number of questions. In this segmentation, all R&D Teams except Colorado and Utah outperformed the control team's take-home in both TPR and TNR. Colorado and Utah outperformed the control team's take-home only in TPR.

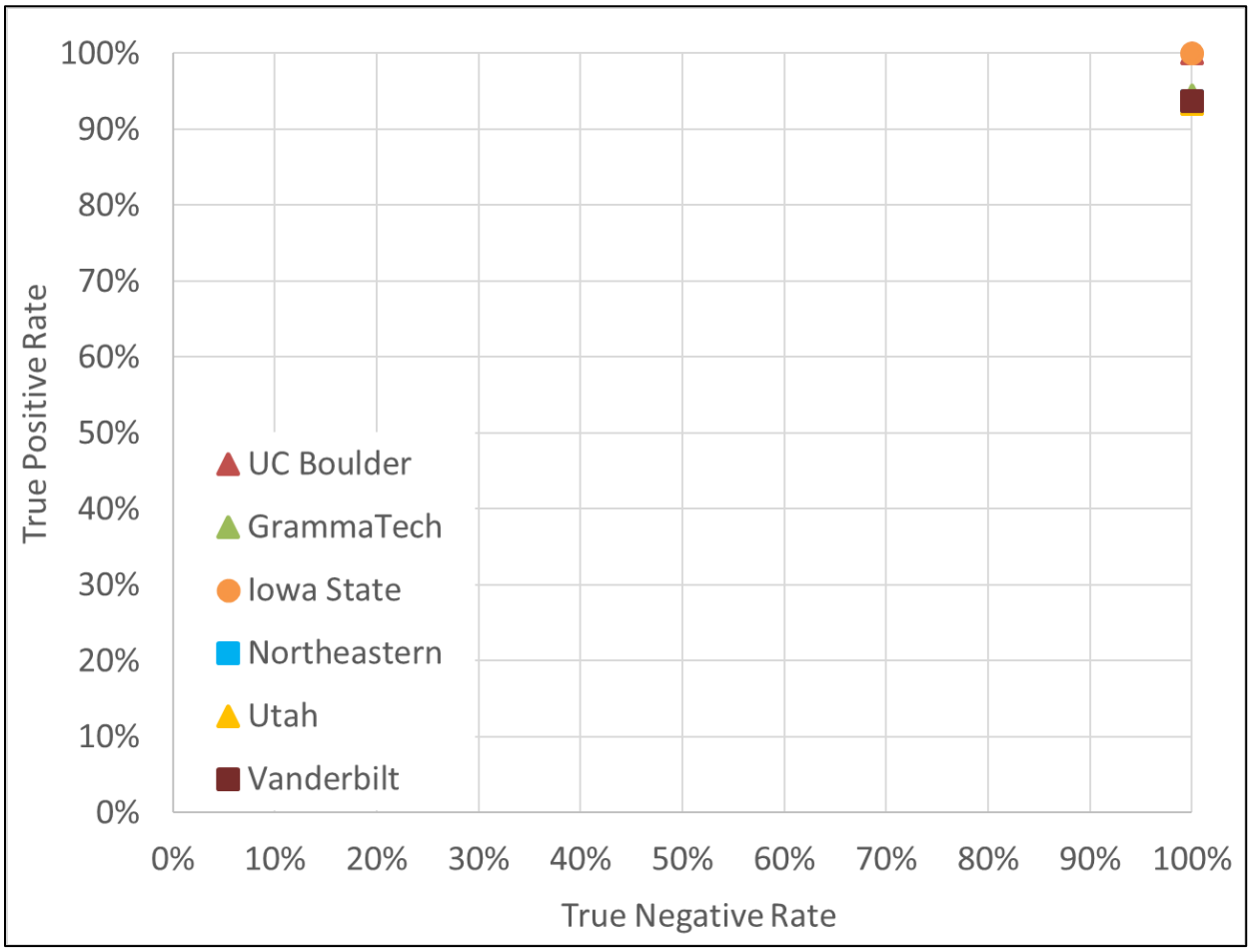


Figure 72: E6 live-collaborative TPR vs TNR segmentation of answered questions

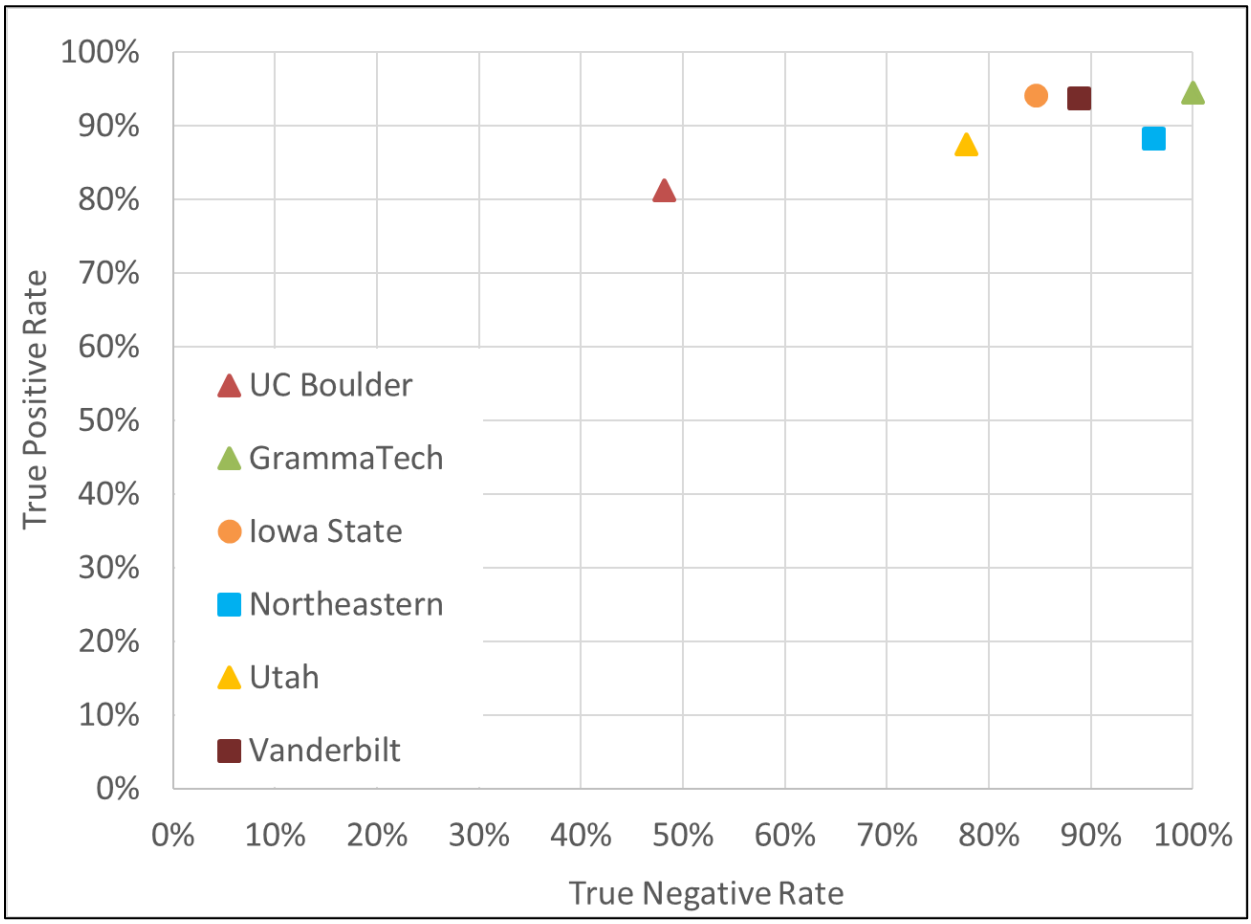


Figure 73: E6 live-collaborative TPR vs TNR segmentation of all questions

4.6.2.1.3 Collaborative Impact Analysis

In Engagement 5, the performance of the Blue Teams in the collaborative engagement motivated a change to focus on side channel questions for Engagement 6. Engagement 6, as a result, contained 42% AC questions and 58% SC questions; however, 14 of the 15 vulnerable (intended and unintended) challenge questions were AC questions. The Blue Teams achieved a much higher accuracy on SC questions (85%) than on AC questions (50%). Following the collaborative engagement, R&D Teams achieved a 93% or higher in all question categories, with 99% on AC questions and 98% on SC questions.

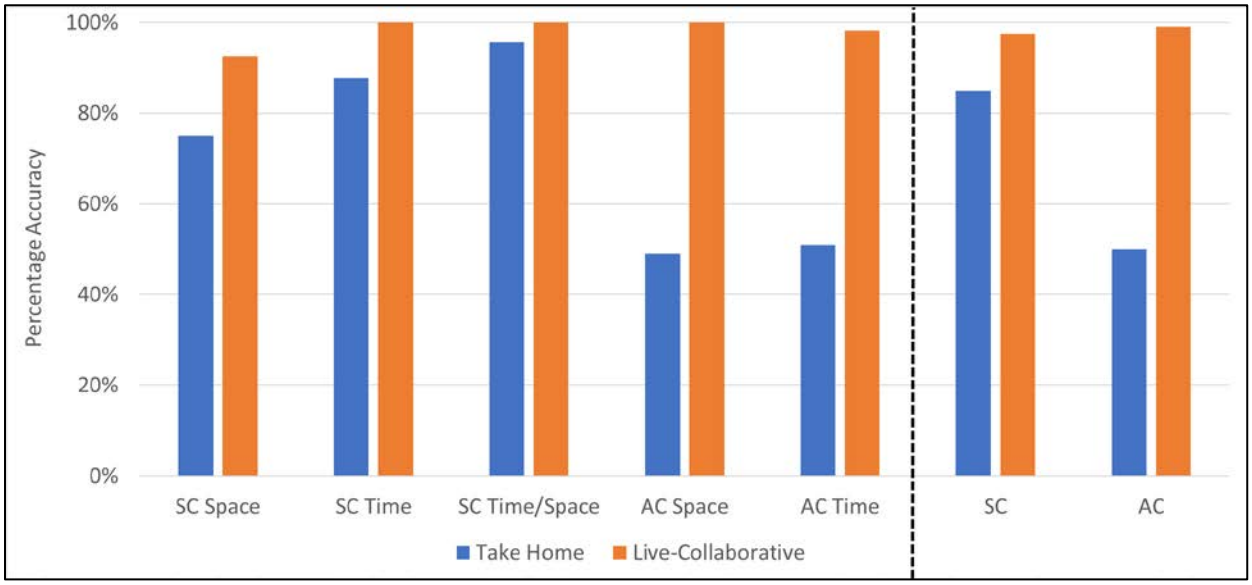


Figure 74: Live collaborative accuracy versus take-home accuracy across challenge question categories

To analyze the impact of different teams during the collaborative engagement, we use a set of metrics based on the performance of a Blue Team's collaborators during the live-collaborative engagement. Performance is measured based on a score, +1 for each correct response, -1 for each incorrect. The first metric looks at the total score change of all collaborators for a given team from the take-home to the live-collaborative engagement. With this first metric, a team can get a question incorrect and yet benefit from fellow collaborators getting this question correct or a team can get a question correct and yet be penalized for fellow collaborators getting this incorrect. This metric is based on the expectation that an impactful team will influence fellow collaborators answer a question correctly (larger score change) and a less impactful team will either fail to convince fellow collaborators or convince them to answer incorrectly.

A second metric looks at the total score change, but only for the collaborators with the same live-collaborative response as a given team. This ensures that a team only benefits from getting the correct final response and convincing others of that correct response.

The third metric looks at the total score change, for collaborators whose live collaborative responses on a given question match the take-home response of the team in question. The difficulty with this metric is that the first two metrics are based on the live-collaborative responses and this metric is based on the take-home responses. These two datasets are not directly comparable as these metrics are driven by the underlying accuracy in the engagement component and the accuracy on the live-collaborative component (98%) is far higher than the accuracy for the take-home component (69%). To allow for direct comparison, the resulting impact scores are normalized by the accuracy of the relevant engagement components. The results are shown below.

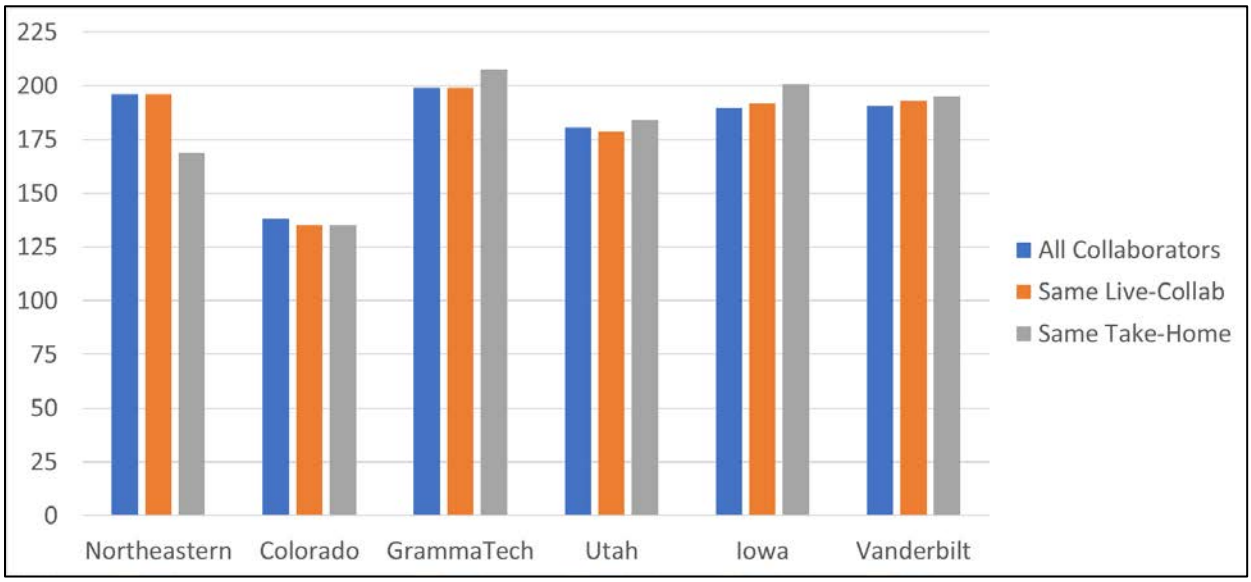


Figure 75: Normalized collaborative impact scores

In all three metrics, GrammaTech had the highest impact. As all teams achieved a near 100% accuracy following the live-collaborative engagement, it is difficult to draw any further conclusions from these metrics beyond a relative scaling of the number of questions a team answered.

Between the take-home and the live collaborative engagement, Utah had the highest increase in accuracy of 42 percentage-points.

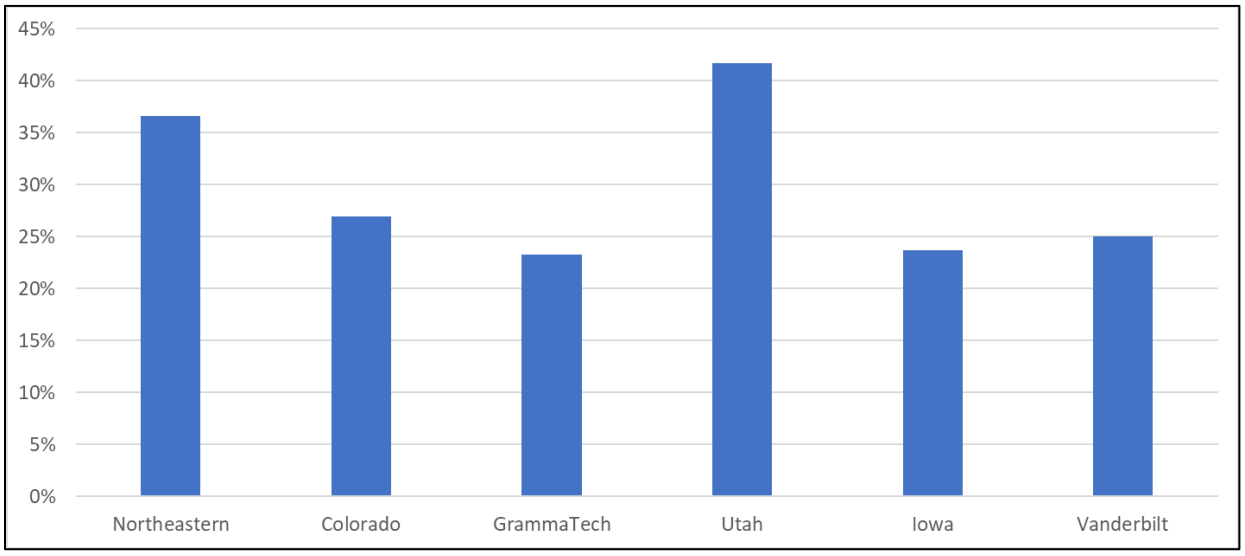


Figure 76: Accuracy increase from take-home to live-collaborative engagement

4.6.2.1.4 Blue Team Responses

The following tables show the Engagement 6 responses for each team. Correct answers are in green, incorrect ones are in red, and unanswered questions are in yellow.

Program	Question	Type	Vuln?	Northeastern	UC Boulder	GrammaTech	Utah	Iowa State	Vanderbilt	Two Six Labs
battleboats_3	Question_022	AC Space	Y	N	Y	Y	N	N	N	Y
battleboats_3	Question_026	SC Space	N	N	0	N	N	N	N	N
battleboats_3	Question_040	SC Time	N	Y	0	N	N	N	N	N
battleboats_3	Question_043	SC Time	N	Y	N	N	N	N	N	N
braidit_3	Question_004	AC Space	Y	Y	0	N	Y	Y	Y	Y
braidit_3	Question_027	SC Space	N	N	N	N	N	N	N	N
braidit_3	Question_029	SC Time	N	N	N	N	Y	0	0	N
braidit_4	Question_032	SC Space	N	Y	Y	Y	0	N	Y	N
calculator_1	Question_014	AC Space	Y	Y	Y	N	N	Y	N	N
calculator_1	Question_016	AC Time	Y	Y	Y	Y	N	Y	Y	Y
calculator_2	Question_003	AC Space	N	N	N	N	N	N	N	N
calculator_2	Question_005	AC Time	Y	Y	Y	Y	N	Y	Y	Y
calculator_2	Question_009	AC Space	Y	Y	N	N	N	Y	N	Y
casedb	Question_019	SC Space	Y	N	0	N	0	0	N	N
casedb	Question_021	SC Time	N	Y	0	N	0	0	N	N
casedb	Question_036	SC Time	N	N	0	N	0	0	N	N
chessmaster	Question_006	SC Time	N	Y	N	N	N	N	Y	N
chessmaster	Question_015	AC Time	Y	N	Y	Y	Y	N	Y	Y
class_scheduler	Question_012	SC Time	N	N	0	N	Y	N	N	N
class_scheduler	Question_017	AC Time	Y	N	Y	N	Y	Y	N	N
class_scheduler	Question_028	SC Time	N	N	0	N	N	N	N	N
effectshero	Question_010	SC Time	N	N	0	N	N	N	N	N
effectshero	Question_025	AC Time	N	N	Y	Y	Y	Y	N	N
railyard	Question_013	AC Space	Y	N	N	N	Y	N	N	N
railyard	Question_018	AC Space	Y	N	Y	Y	Y	Y	Y	Y
railyard	Question_035	SC Space	N	N	0	N	N	N	N	N
simplevote_3	Question_001	SC Time/Space	N	N	Y	N	N	N	N	N
simplevote_3	Question_033	SC Time	N	N	0	N	N	N	N	N
simplevote_3	Question_034	SC Space	N	Y	N	N	N	N	N	N
simplevote_3	Question_037	SC Space	N	Y	Y	Y	N	N	N	N
simplevote_3	Question_038	AC Time	Y	Y	N	N	Y	N	N	0
simplevote_4	Question_023	SC Time/Space	N	N	N	N	0	N	N	0
simplevote_4	Question_030	AC Time	Y	Y	Y	Y	Y	N	N	Y
simplevote_4	Question_039	SC Space	N	N	0	N	0	N	N	N
simplevote_4	Question_041	SC Time	N	N	0	N	0	N	N	N
stac_coin	Question_007	AC Time	N	N	0	N	N	Y	N	0
stac_coin	Question_008	AC Space	N	Y	N	N	N	N	N	0
stac_coin	Question_042	SC Time/Space	N	N	0	Y	N	0	0	0
swappigans	Question_002	AC Time	Y	N	Y	Y	Y	Y	Y	Y
swappigans	Question_020	SC Time/Space	N	Y	N	Y	Y	Y	Y	Y
swappigans	Question_031	SC Time	N	N	N	N	N	N	N	N
tollbooth	Question_011	AC Time	Y	0	0	Y	N	N	N	Y
tollbooth	Question_024	SC Time/Space	N	0	0	N	N	N	0	Y

Figure 77: E6 take-home responses (red – incorrect, green – correct, yellow – no response)

Figure 78 below shows the Blue Teams responses updated for the live-collaborative engagement. Following the collaborative engagement, CaseDB question 19 remains the only engagement question that the R&D Teams did not collectively answer correctly.

Program	Question	Type	Vuln?	Northeastern	UC Boulder	GrammaTech	Utah	Iowa State	Vanderbilt	Two Six Labs
battleboats_3	Question_022	AC Space	Y	Y	Y	Y	Y	Y	Y	Y
battleboats_3	Question_026	SC Space	N	N	0	N	N	N	N	N
battleboats_3	Question_040	SC Time	N	N	0	N	N	N	N	N
battleboats_3	Question_043	SC Time	N	N	N	N	N	N	N	N
braidit_3	Question_004	AC Space	Y	Y	0	Y	Y	Y	Y	Y
braidit_3	Question_027	SC Space	N	N	N	N	N	N	N	N
braidit_3	Question_029	SC Time	N	N	N	N	N	0	0	N
braidit_4	Question_032	SC Space	N	Y	Y	Y	0	Y	Y	N
calculator_1	Question_014	AC Space	Y	Y	Y	Y	Y	Y	Y	N
calculator_1	Question_016	AC Time	Y	Y	Y	Y	Y	Y	Y	Y
calculator_2	Question_003	AC Space	N	N	N	N	N	N	N	N
calculator_2	Question_005	AC Time	Y	Y	Y	Y	Y	Y	Y	Y
calculator_2	Question_009	AC Space	Y	Y	Y	Y	Y	Y	Y	Y
casedb	Question_019	SC Space	Y	N	0	N	0	0	N	N
casedb	Question_021	SC Time	N	N	0	N	0	0	N	N
casedb	Question_036	SC Time	N	N	0	N	0	0	N	N
chessmaster	Question_006	SC Time	N	N	N	N	N	N	N	N
chessmaster	Question_015	AC Time	Y	Y	Y	Y	Y	Y	Y	Y
class_scheduler	Question_012	SC Time	N	N	0	N	N	N	N	N
class_scheduler	Question_017	AC Time	Y	Y	Y	Y	Y	Y	Y	N
class_scheduler	Question_028	SC Time	N	N	0	N	N	N	N	N
effectshero	Question_010	SC Time	N	N	0	N	N	N	N	N
effectshero	Question_025	AC Time	N	N	N	N	N	N	N	N
railyard	Question_013	AC Space	Y	Y	Y	Y	Y	Y	Y	N
railyard	Question_018	AC Space	Y	Y	Y	Y	Y	Y	Y	Y
railyard	Question_035	SC Space	N	N	0	N	N	N	N	N
simplevote_3	Question_001	SC Time/Space	N	N	N	N	N	N	N	N
simplevote_3	Question_033	SC Time	N	N	0	N	N	N	N	N
simplevote_3	Question_034	SC Space	N	N	N	N	N	N	N	N
simplevote_3	Question_037	SC Space	N	N	N	N	N	N	N	N
simplevote_3	Question_038	AC Time	Y	Y	Y	Y	Y	Y	Y	0
simplevote_4	Question_023	SC Time/Space	N	N	N	N	0	N	N	0
simplevote_4	Question_030	AC Time	Y	Y	Y	Y	Y	Y	Y	Y
simplevote_4	Question_039	SC Space	N	N	0	N	0	N	N	N
simplevote_4	Question_041	SC Time	N	N	0	N	0	N	N	N
stac_coin	Question_007	AC Time	N	N	0	N	N	N	N	0
stac_coin	Question_008	AC Space	N	N	N	N	N	N	N	0
stac_coin	Question_042	SC Time/Space	N	N	0	Y	N	0	0	0
swappigans	Question_002	AC Time	Y	Y	Y	Y	Y	Y	Y	Y
swappigans	Question_020	SC Time/Space	N	Y	N	Y	Y	Y	Y	Y
swappigans	Question_031	SC Time	N	N	N	N	N	N	N	N
tollbooth	Question_011	AC Time	Y	0	0	Y	N	Y	Y	Y
tollbooth	Question_024	SC Time/Space	N	0	0	N	N	N	0	Y

Figure 78: E6 take-home responses (red – incorrect, green – correct, yellow – no response)

4.6.2.1.5 Categories of Incorrect Responses

Incorrect responses by Blue Teams fall into four categories. These categories are not identical for AC and SC questions, but they mirror each other. For AC questions, the incorrect Blue Team response categories are:

AC 9. Did not follow directions or violated the question definitions
AC 10. Failed to identify the complexity within the challenge program
AC 11. Failed to understand the input to the complexity control flow of the challenge program
AC 12. Failed to properly determine the complexity bounds of an identified complexity

For SC questions, the incorrect Blue Team response categories are:

SC 9. Did not follow directions or violated the question definitions
SC 10. Failed to identify the secret or the secret location within the challenge program
SC 11. Failed to identify the side channel (processing conditioned on the secret value)
SC 12. Failed to properly analyze the side channel strength

An incorrect response can fit into multiple categories e.g. given strong side channel a team identifies a weaker side channel. This is classified as both a failure to identify the intended side channel vulnerability and a failure to correctly determine strength.

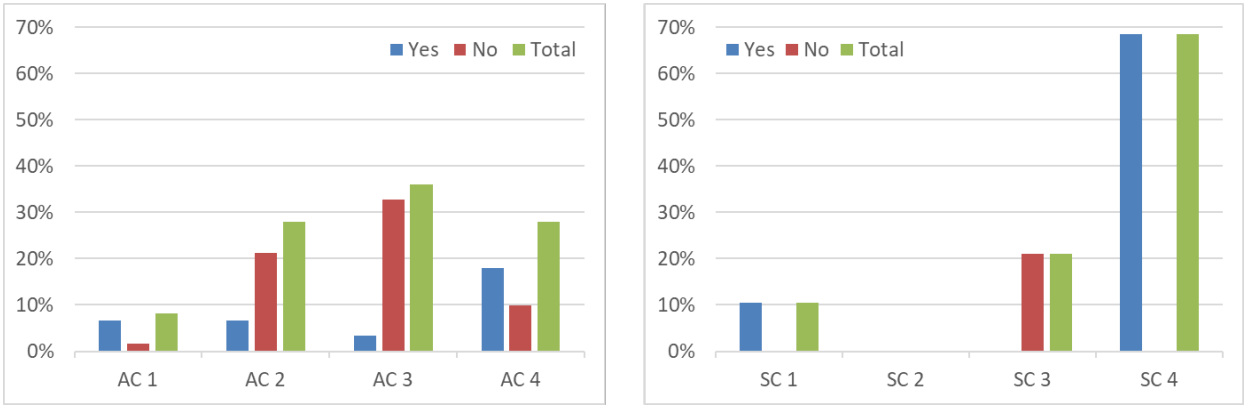


Figure 79: E6 Take-home categories of incorrect responses; algorithmic complexity (left); side channel (right)

The data shows that for AC questions, failure to understand the input to the complexity was the primary reason for incorrect responses (36%) followed closely by failure to identify the complexity and failure to determine strength. Failure to identify complexity and failure to understand complexity input, caused teams to incorrectly declare challenges non-vulnerable. 8% of incorrect AC responses were due to a failure to follow directions/violation of question definitions.

For SC questions, the dominant reason for incorrect responses (68%) was a failure to analyze strength, resulting in teams incorrectly declaring challenge programs vulnerable. In a distant second, 21% of incorrect SC responses were due to a failure to identify the side channel, motivating teams to incorrectly declare challenge programs non-vulnerable. The SC space question for the Case DB challenge was the major cause of the cases of failure to identify the side channel (SC 3) due to the complex distributed nature of the vulnerability.

4.6.2.2 AC Teams

4.6.2.2.1 CyberPoint

CyberPoint provided 7 challenge programs resulting in 22 challenge questions with 3 intended vulnerable and 19 intended non-vulnerable. Blue teams discovered unintended vulnerabilities that impacted 5 intended non-vulnerable CyberPoint challenge questions. Blue teams provided 18 more responses to CyberPoint questions than to BBN questions (140 versus 122) and achieved a higher accuracy on CyberPoint questions than on BBN questions (74% versus 69%).

4.6.2.2.1.1 Unintended Vulnerabilities

4.6.2.2.1.1.1 BraidIt 3 Question 004 Intended Null AC Space

During the take-home Two Six Labs and Iowa found braids they could send to the server which would cause an infinite loop when the original and modified braids are compared. Messages were stored in log files at each step of this loop, causing the log files to almost double the resource usage limit. Two Six Lab's malicious modified braid is "NOPQRSTUVWXYZZZZZZZZZZZZZZZZZZZZZZZZZZnmopqrstuvwxC" and as long as the original braid being compared against doesn't contain the letters "a", "b" or "c", the loop will be infinite and cause the log file to continue writing more lines. Iowa's modified braid is "𝘈𝘛ʂʐɐɑɒqɦɬɱɻ𝘈𝘛𝘛" and as long as each of the original 5 braids is "yyy", the process of checking if the braids are equal causes an infinite loop resulting in 40 MB of log files, exceeding the target of 25 MB.

4.6.2.2.1.1.2 Calculator 1 Question 014 and Calculator 2 Question 009 Intended Null AC Space

Colorado and Iowa found that very large numbers (ex. 999^1695) multiplied together as many times as the input budget allowed causes the resource usage limit to be exceeded. Northeastern discovered that repeatedly dividing a large number (ex. 10^499) by 1 consumes all available memory after 400 divisions.

4.6.2.2.1.1.3 Calculator 1 Question 016 Intended Null AC Time

Vanderbilt discovered that input of the form "$(E)(E)(E)$ ..." with $E = 999\text{^}1695 * 999\text{^}1695 * 999\text{^}1695 * 999\text{^}1695 * 999\text{^}1695$ repeated 212 times (maximum allowed by budget) causes the benign request to be delayed by more than 120 seconds. Colorado found that multiplying very large numbers against each other as many times as the budget allows causes the benign request to exceed the resource usage limit. The specific input: $\overbrace{A \ldots A}^{120}$, where A is $(999^{1666} * 999^{1666} * 999^{1666} * 999^{1666}) * (999^{1666} * 999^{1666} * 999^{1666} * 999^{1666} * 999^{1666})$.

4.6.2.2.1.1.4 SimpleVote 3 Question 038 Intended Null AC Time

With help from Northeastern, the R&D Teams discovered that registration keys with large "core" numbers take the hashing algorithm to take longer to complete. To fit the exploit within the input budget, the SSL session must be held open with multiple registration keys submitted within the same session.

4.6.2.2.1.1.5 BraidIt 4 Question 032 Out-of-Scope SC Space

GrammaTech discovered that with an existing tool that predicts the PRNG sequence of `java.util.Random` the randomization used to prevent a vulnerability can be reversed.

4.6.2.2.1.2 Engagement Results

Table 37: CyberPoint take-home results

Challenge Program	Positive Qs	Null Qs	Total Qs	Attempts	Correct Attempts	Percentage Correct
BattleBoats 3	1	3	4	26	20	77%
BraidIt 3	1	2	3	18	13	72%
BraidIt 4	0	1	1	6	3	50%
Calculator 1	2	0	2	14	9	64%
Calculator 2	2	1	3	21	16	76%
SimpleVote 3	1	4	5	33	24	73%
SimpleVote 4	1	3	4	22	19	86%
Total	**8**	**14**	**22**	**140**	**104**	**74%**

Table 38: CyberPoint live-collaborative results

Challenge Program	Positive Qs	Null Qs	Total Qs	Attempts	Correct Attempts	Percentage Correct
BattleBoats 3	1	3	4	26	26	100%
BraidIt 3	1	2	3	18	18	100%
BraidIt 4	0	1	1	6	6	100%
Calculator 1	2	0	2	14	13	93%
Calculator 2	2	1	3	21	21	100%
SimpleVote 3	1	4	5	33	33	100%
SimpleVote 4	1	3	4	22	22	100%
Total	**8**	**14**	**22**	**140**	**139**	**99%**

4.6.2.2.2 Raytheon/BBN Technologies

BBN provided 8 challenge programs resulting in 21 challenge questions with 5 intended vulnerable and 16 intended non-vulnerable. Blue teams discovered unintended vulnerabilities that impacted 2 intended non-vulnerable BBN challenge questions. Blue teams provided 18 more responses to CyberPoint questions than to BBN (140 versus 122) and achieved a higher accuracy on CyberPoint questions than on BBN questions (74% versus 69%).

4.6.2.2.2.1 Unintended Vulnerabilities

4.6.2.2.2.1.1 Class Scheduler Question 017 Intended Null AC Time

During the take-home Colorado identified a broken input guard that allows courses with a negative number of sections. This allows an attacker to provide an arbitrary large number of sections for the required courses, while bypassing the check that total number of courses must be less than 29. During the collaborative engagement, the R&D Teams observed that by using this bug with 5 simultaneous requests, the resource usage limit for the benign request can be exceeded.

4.6.2.2.2.1.2 Railyard Question 013 Intended Null AC Space

During the live-collaborative engagement, the teams identified that by the intended AC Space (disk) vulnerability results in a non-terminating loop that if broken causes an unintended AC Space (memory) vulnerability. They observed that this can be achieved by setting the `Coal_Car`'s static `next` field to `NULL` causing an indefinite write to `printToStream`.

4.6.2.2.2.1.3 Swappigans Question 031 Out-of-Scope SC Time

Blue teams discovered that with knowledge of the target user's username (question-provided assumption) an attacker can break the encryption via brute force.

4.6.2.2.2.2 Engagement Results

Table 39: BBN take-home results

Challenge Program	Positive Qs	Null Qs	Total Qs	Attempts	Correct Attempts	Percentage Correct
CaseDB	1	2	3	12	7	58%
Chessmaster	1	1	2	14	9	64%
Class Scheduler	1	2	3	19	12	63%
Effects Hero	0	2	2	13	9	69%
Railyard	2	1	3	20	11	55%
STAC Coin	0	3	3	14	12	86%
Swappigans	1	2	3	21	19	90%
Tollbooth	1	1	2	9	5	56%
Total	**7**	**14**	**21**	**122**	**84**	**69%**

Table 40: BBN live-collaborative results

Challenge Program	Positive Qs	Null Qs	Total Qs	Attempts	Correct Attempts	Percentage Correct
CaseDB	1	2	3	12	8	67%
Chessmaster	1	1	2	14	14	100%
Class Scheduler	1	2	3	19	18	95%
Effects Hero	0	2	2	13	13	100%
Railyard	2	1	3	20	19	95%
STAC Coin	0	3	3	14	14	100%
Swappigans	1	2	3	21	21	100%
Tollbooth	1	1	2	9	9	100%
Total	**7**	**14**	**21**	**122**	**116**	**95%**

4.7 Engagement 7

4.7.1 Engagement Plan

4.7.1.1 Purpose and Program Scope

4.7.1.1.1 Program Scope

This Engagement Plan is for Engagement 7, a combined take-home and live engagement. The final engagement will be released to the control team on 1/14/2019 and released to the R&D Teams on 1/28/2019. The live engagement will take place from 2/5/2019; answers must be submitted at the end of the engagement on 2/8/2019. Answers should be sent to evan.fortunato@apogee-research.com and kwame.ampeh@apogee-research.com or uploaded to Apogee's Gitlab server.

The document is structured into the following sections.

Section 1: Purpose and Program Scope
Section 2: Operational Definitions and Research Hypotheses
Section 3: Engagement 3 Plans and Logistics
Section 4: Post-Engagement Analysis Plans

4.7.1.2 Operational Definitions and Program Hypotheses

4.7.1.2.1 Operational Definitions of STAC Vulnerabilities

To guide the Engagements, the STAC AC, EL, and government teams have come up with operational definitions of vulnerability to algorithmic complexity attack and to side channels. The AC performers will use these definitions to determine when the vulnerabilities they've inserted into their challenge programs are strong enough for use in engagements. The Control and R&D Teams will use these operational definitions to determine when they've found a vulnerability worth reporting – i.e., if it meets the definition of a STAC vulnerability. The operational definition document, currently *2017-08-22-opdef-v08.1*, specifies the definitions for STAC vulnerabilities along with guidance for Blue team responses to engagement challenges.

4.7.1.2.1.1 Unintended Vulnerabilities in Challenge Programs

Many Algorithmic Complexity and Side Channel vulnerabilities may exist in each challenge program beyond those intentionally inserted. For example, a normal authentication function is technically a side channel that leaks a secret. However, most of these vulnerabilities are "weak" – e.g., for Algorithmic Complexity, the additional complexity is a small percentage of the normal complexity, and for Side Channels, the number of operations necessary to resolve the secret is impractically large.

The operational definitions defined above are designed to minimize these unintended vulnerabilities in the challenge programs with the use of strong vulnerability thresholds (e.g., adversary operation budget is small for Side Channels). These thresholds will serve to discount most of these "unintended" vulnerabilities as not meeting the STAC definition. However, some unintended vulnerabilities will occur that do meet the definition. A reported vulnerability that satisfies the operational definition criterion for a given challenge program is a correct response, regardless if it is an intended or unintended vulnerability on behalf of the AC teams. During the

analysis period after each engagement, the EL team will work with the Blue teams to verify that reported unintended vulnerabilities do actually meet the operational definition so that full credit is given. Unintended vulnerabilities can fall into two categories: those "in-scope" as per the STAC Operational Definitions document and those "out-of-scope" as per the STAC Operational Definitions document. Full credit will be given for all verified unintended vulnerability, whether they are "in-scope" or "out-of-scope." In the case of "in-scope" unintended vulnerabilities, the engagement answer key will be updated to reflect the identified unintended vulnerabilities.

4.7.1.2.2 Research Hypothesis and Segmentation of Challenge Problems

Throughout the STAC program, the EL team will have a set of research hypotheses pertaining to performance of the tools with respect to different types of STAC vulnerabilities and challenge problems.

Example hypotheses include:

19. Complexity vulnerabilities can be due to: (1) a change in the complexity of the algorithm, (2) a change in the coefficients of a fixed complexity, or (3) a flawed data guard that allows processing of input data that shouldn't be allowed. How do the different R&D techniques perform with respect to each of these types of complexity vulnerabilities?
20. Side channels can be weakened by increasing the number of secret symbols that are mapped to identical observed symbols or by increasing the width of the parent distributions of the observed symbols (causing overlaps of the of the samples from different observed symbol distributions). How do the different R&D techniques perform with respect to each of these challenges?
21. Some non-vulnerable programs are locally balanced (e.g., all paths through a sub-section of the binary consume equivalent time and space resources) while others are non-locally balanced (e.g., resource consumption is asymmetric on a set of paths in multiple parts of the program resulting in entire execution traces that are balanced). How do the different R&D techniques perform with respect to the locality of balancing of resource consumption?

In support of these hypotheses, each challenge problem will be assessed and categorized with respect to how results on the challenge problem will support resolution of each of the hypotheses. This segmentation of the challenge problems will not be exposed during the engagements, but will be tracked for post-engagement analysis.

During the engagements, each challenge problem will come with questions such as:

25) Does the challenge problem contain a time-based side channel?
26) Does the challenge problem contain a space-based side channel?
27) Does the challenge problem contain a time-based complexity vulnerability?
28) Does the challenge problem contain a space-based complexity vulnerability?

Note, that the questions will be more specific version of these classes of questions and include additional context information and could include questions regarding combined time and space vulnerabilities.

4.7.1.3 Common Reference Platform

Although STAC focuses on the analysis of Java programs, reasoning about these programs under the context of the defined vulnerability operation definitions (Section 2) requires an

understanding of the resource usage of operations occurring in the Java runtime environment, operating system, and underlying hardware. The resource usage may vary between different hardware/software platforms, making it difficult to define platform independent resource limits and side channel vulnerabilities respecting the same operation budget on all platforms.

To mitigate this, the AC teams will describe challenge program vulnerabilities in terms of a common reference platform. The Control and R&D teams are not required to perform their analysis on this common platform. However, the EL will evaluate the results in terms of the vulnerabilities that exist under the common platform. Thus, Control and R&D teams using different platforms in their analysis must manage whatever resource limit and operation budget translation is necessary to make the results relevant to the common platform.

Docker will be used for easy access to Challenge Problems. The EL team has run a series of experiments with Docker (see the *STAC-EL Reference Platform Analysis v2.0* document) to determine any difference between the resource usage in Docker and bare metal. Docker has no impact on space usage; therefore, the EL expects no impact on space-based vulnerabilities. Small differences are seen in the timing results on Docker. Given that the time difference is very small, the EL expects it to have no impact on algorithmic complexity vulnerabilities; however, these small differences may have a small impact on the strength of time-based side channels. The EL will carefully evaluate each challenge program to ensure that side channels are properly detectable at a sufficient strength to meet the operational definition. The EL will continue to explore these differences and look at ways to minimize the impact of the use of Docker while still getting its advantage of separating challenge problems to ensure that there are no conflicts. Any questions about time or space usage will be resolved via a clean install on bare metal of the reference platform.

For Engagement 7, the common reference platform is specified in the *STAC-EL Reference Platform v3.2.3* document.

4.7.1.3.1 AC Team Deliveries

AC Teams should deliver final versions of their challenge programs for Engagement 7 into their GitLab repository by 10/10/2018. Deliveries should be structured in the following way:
Each Challenge Program should have the following directory structure:

- Blue Team Information in a `./<challenge id>/BT/` folder:
  - `description.txt` should be a text file describing how to execute the Challenge Program – in a form similar to a man-page.
  - `APIsUsed.txt` should be a text file that provides a list of the external APIs used in this challenge program.
  - `challenge_program/` sub-folder should contain a tar file containing the Challenge Program jar file and any supporting files required for a TA-1 team to execute the Challenge Program on the STAC reference platform (`NUC` with `CENTOS 7` and `OpenJDK` run time environment installed). This folder will also contain an example input.
  - `questions/` this sub-folder is optionally populated with any draft questions that the AC team think should be included. The EL team will heavily edit or augment this information before the engagement. Each question file will contain a single question (one of the four classes described in Section 2.2) with supporting context.

- EL Team Information in a `./<challenge id>/EL/` folder:
  - `fulldescription.txt` should be a text file describing the malice contained in the challenge program in simple English. It should include specific information on the malice such as how the secret is stored and how it is leaked via a side channel.
  - `source/` sub-folder should contain the source code for the Challenge Program.
  - `build/` sub-folder should contain a script named ./`create_cp.sh`, which performs the following operations:
    - Compiles the Challenge Program from `./../source/`
    - Creates the tar file described above for the `./../../BT/challenge_program/` sub-folder.
    - Creates a file named ./`Dockerfile` with configuration instructions for the layers of a Challenge Program `docker` image – based upon a base image `stac/base:<version>` containing `Centos 7` and `OpenJDK`; example:

```
FROM stac/base:v1
ADD encrypt_header.tar /home/
```

  - `budgets/` sub-folder should contain information about the recommended adversary budgets for each of the malice contained in the Challenge Program. The EL team will cross-check and augment this information before crafting the final version of questions. In addition, it should include any constraints to make the Challenge Program practical (e.g. "*password limited to no more than 30 alphanumeric characters*").
  - `proofs/` sub-folder should contain one sub-folder for each malice included in the Challenge Program. It should contain material that provides clear evidence that the Challenge Program includes the specific vulnerability (within the Operational Budget) asked about by the question. Each folder should be named.

If either AC team prefers, a variation of the above format is acceptable if it contains the relevant information and the format is common for all deliveries by that AC team.

4.7.1.3.2 Challenge Problems during the Engagement

During an engagement, the Blue teams will be presented with a summary of the Challenge Problems and Questions and an (encrypted) Challenge Docker file (password provided separately). The Docker file (and supporting script) will provide a way to instantiate each of the challenge problems with the information described in Section 4.7.1.3.1 for the BT directory. In addition to the Docker file, the EL will provide a tar ball of the challenge programs and challenge questions. This will allow teams easy access to the information, noting that the challenge programs are only confirmed to run properly via the Docker registry.

For each Challenge Program, Blue teams will need to supply a "Yes" or "No" answer with supporting justification that describes the vulnerabilities and why it exceeds the threshold defined in the operation definition. Even if the correct "Yes"/ "No" answer is provided, if the justification does not reasonably align with a present vulnerability (intended or unintended), the answer will be scored as incorrect. The EL team will accept a wide range of justifications and a proof of vulnerability is not required (especially for noisy side channels, where the proof of vulnerability might be quite complex). Further guidance for Blue team engagement responses can be found in the operational definition document, *2017-08-22-opdef-v08.1*.

4.7.1.4 Engagement 7 Details

4.7.1.4.1 Overview

Engagement 7 is a combined take-home and live engagement. The goal of this final engagement is to compare the combined performance of all R&D Teams to the performance of the control team. Only the R&D Teams will collaborate during the live engagement. The control team will be present but will continue to work on the challenges without collaborating with the R&D Teams. **Research and control teams should answer all questions in the engagement.**

For the Research teams, the take-home engagement will start on 1/28/2019 at noon Eastern time. As the combined R&D Teams will have more people than the control team, the control team will have additional time in the take-home to equalize the total effort. Therefore, the Control team will get the take-home engagement on 1/14/2019 at noon Eastern time.

During the take-home component of the engagement, Blue Teams are expected to process and begin analysis of the challenge programs and challenge questions. Unlike previous combined take-home and live engagements, Blue Teams are not expected to submit answers following the take-home component of the engagement. By 11:59PM Sunday, 2/3/2019 all Research teams must submit their status reports (in one of the official formats) which includes:

1) A list of the four people that will be attending the live engagement and a designation of one of the four people as the team lead for administrative / coordination issues. It is not expected that additional people will be approved for any of the R&D Teams, but if a team would like to bring additional staff, please submit a request to Apogee Research and DARPA for consideration no later than November 30th, 2018.
2) A current status of each question (see definition below) and the names of two people that can collaborate on that question. If only 1 person can viably represent on a particular question, please note that, it is requested that this happen only rarely as it will interfere with efficient scheduling of the live event.

The R&D Team status on each challenge question will be classified in the following categories:

1. Ran and understood the functionality of the challenge program. Read and understood the challenge question.
2. Tools ran successfully on the challenge program, generated meaningful output, but not enough confidence to answer.
3. Tools ran successfully on the challenge program, generated meaningful output, have enough confidence to answer, provide the answer.

The collaborations for the R&D Teams will be ranked and scheduled using the self-reported challenge question statuses and the staff available to represent on each question. To facilitate collaboration, the status submitted by each R&D Team will be made available to all R&D Teams (but not the control team) at the collaborative engagement. During the live-collaborative engagement, R&D Teams are expected to collaborate, but are not required to reach consensus after collaboration.

Following the live engagement, both R&D Teams and the control team are expected to submit answers to all challenge questions. No answers are expected prior to the completion of the live event.

The PI meeting (2/4/2019) will take place before the live collaborative engagement. During the PI meeting, R&D Teams should demonstrate a run of their tools on a previous engagement challenge. Each R&D Team should be prepared to have a member of the audience run the demonstration with guidance from the team.

4.7.1.4.2 Clarifications

4.7.1.4.2.1 Budget Ambiguity

We have identified ambiguities in past engagements in our definitions of what the attacker has prior access to analyze offline and the attacker's budget to build a database to map observable symbols to secret symbols. We plan to address these in engagement 7 with the following clarifications:

1. The notes section of the description.txt for each challenge program will state what the attacker has prior access to analyze offline (e.g. all the source code and certain canned public data files), and what the attacker doesn't have prior access to analyze offline (e.g. server private key data files and user authentication data files).
2. Any information not contained within the set of items that the attacker has prior access to analyze offline must be extracted via active operations, passive operations, or notional oracle queries at the costs defined in the challenge questions.
3. Unless otherwise stated in the description or question, the challenge start scripts (noted in the "Engagement Tools" section) run on the reference platform/network environment establish the state of the application prior to attack.

4.7.1.4.2.2 Direction of Information Leakage

In the analysis of challenge programs for information leakage, information can be leaked in any traffic between the server and the client (serverNuc and clientNuc traffic captured on the mirrored data device on masterNuc) regardless of the direction of the flow of traffic. Note: the previous guidance on scope still applies.

4.7.1.4.3 Schedule and Logistics

4.7.1.4.3.1 Schedule

The schedule below provides an overview of the milestones for Engagement 7, including the engagement itself and the post engagement evaluation.

- Take-home engagement: Starts at noon Eastern on 1/28/2019 for the R&D Teams and 1/14/2019 for the Control Team. All teams will submit their status by 11:59PM on Sunday, February 3rd, 2019.
- PI Meeting: 2/4/2019 at TBD.
- Live collaborative engagement: 2/5/2019 to 2/8/2019 at TBD.
- Release Engagement Plans (EL team): Initial release on 10/10/2018. Post on GitLab Public_EL_Information Project.
- AC Teams deliver completed versions of all engagement challenges for review to the EL: 10/10/2018 by noon eastern time. (Does not apply in the case of previously discussed exceptions).
- AC Teams deliver final versions of all Engagement 7 challenges to the EL: 11/6/2018 by noon eastern time. (Does not apply in the case of previously discussed exceptions)

- The EL team releases the take-home Engagement 7 challenges to Blue teams on 1/28/2019 (1/14/2019 for the Control Team).
- Quick-look engagement results (EL team): initial quick-look results presented to DARPA after the live collaborative engagement on 2/15/2019. Reviewed results subsequently posted to GitLab by 3/8/2019.
- Final Engagement 7 analysis report delivered to DARPA: 4/26/2019.

4.7.1.4.3.2 Logistics

The 2/4/2019 PI meeting and the Engagement 7 live collaborative engagement will take place at TBD. The live collaborative portion of Engagement 7 will occur from 2/5/2019 through 2/8/2019. The detailed schedule is:

- Monday, 2/4/2019:
  - 7:00 AM: EL team arrives for setup.
  - 7:30 AM – 8:00 AM: STAC teams arrive for PI meeting.
  - 8:00 AM – 5:30 PM: PI meeting.
  - 5:30 PM – 7:00 PM: Additional Setup time. No overnight processing.
- Tuesday, 2/5/2019: Day 1:
  - 8:30 AM: Engagement starts for all teams.
  - Noon: Everyone pauses engagement for lunch.
  - 1:00 PM: Engagement restarts for all teams.
  - 4:30 PM: Day 1 engagement ends. Teams are welcome to leave processes running overnight, but no work should be done by the teams on any part of the problem (honor system).
- Wednesday, 2/6/2019: Day 2:
  - 8:30 AM: Engagement starts for all teams.
  - Noon: Everyone pauses engagement for lunch.
  - 1:00 PM: Engagement restarts for all teams.
  - 4:30 PM: Day 2 engagement ends. Teams are welcome to leave processes running overnight, but no work should be done by the teams on any part of the problem (honor system).
- Thursday, 2/7/2019: Day 3:
  - 8:30 AM: Engagement starts for all teams.
  - Noon: Everyone pauses engagement for lunch.
  - 1:00 PM: Engagement restarts for all teams.
  - 4:30 PM: Day 3 engagement ends. Teams are welcome to leave processes running overnight, but no work should be done by the teams on any part of the problem (honor system).
- Friday, 2/8/2019: Day 4:
  - 8:30 AM: Engagement restarts for all teams.
  - 1PM: Day 4 engagement ends.

Each R&D Team should bring 4 people to the live engagement. The expected number of people for the control team will be discussed with the control team prior to the engagement.

If teams would like to do remote processing, they are welcome to under the following restrictions and understandings:

- While it is the intent to provide external connectivity, we cannot guarantee high quality

of service connectivity during the event (dependent on engagement location infrastructure). If connectivity is required for your approach to work, please have backup plans available.

- No one not at the engagement should work on, touch or be involved with the analysis effort in any way during the engagement.

4.7.1.5 Post Engagement Analysis Plan

Engagement 7 is a combined take-home prep and live collaborative engagement. Blue teams should answer all challenge questions. The take-home portion of the engagement allows both R&D Teams and the control team to understand the challenge programs and challenge questions, and to run their tools and perform analysis on the engagement challenges. Following the take-home, Blue Teams are not expected to submit any answers to the engagement challenges.

During the live component of the engagement, R&D Teams are expected to collaborate similar to previous live-collaborative engagements, while the control team will continue to work without collaborating with other teams. As stated in section 4, following the take-home, R&D Teams are expected to submit the state of the analysis performed on each challenge question, and the names of at least two people capable of collaborating on each challenge question. This information will be used to prioritize scheduling of challenge questions in the collaborative engagement. Following collaborations, R&D Teams are not expected to reach consensus on answers. Each team will be evaluated individually on the answers submitted after the live-collaborative engagement. In addition, the combined performances of the R&D Teams will be compared to the performance of the control team to help assess the research hypothesis of the STAC program.

4.7.1.5.1 Metrics

The following set of metrics will be used to evaluate Blue team performance in Engagement 7:

- True Positive Rate (Number of correct “Yes” responses/ Total number of vulnerable challenge questions)
- True Negative Rate (Number of correct “No” responses/ Total number of non-vulnerable challenge questions).

Note, True Positive Rate is equivalent to a Probability of Detection Metric while the True Negative Rate is equal to 1 – False Alarm Rate. Using the True Positive and True Negative Rate gives insight into the relative strengths of each Blue team with respect to finding and ruling out vulnerabilities but doesn’t give insight into the speed of the Blue teams.

For each Blue team, the EL will evaluate the justifications against the justification guidance provided. If the justification is sufficient, then the answer will be considered a justified answer and will be included in the justified answer analysis. As Blue Teams are expected to answer all challenge questions, properly justified answers will be considered (in the numerator) and all challenge questions will be considered (in the denominator). This segmentation shows the benefits of speed [and breadth] as Blue teams that only answer a small fraction of the questions will not be able to score well under the second segmentation.

To achieve the segmentation, each Blue team’s answers will be evaluated against the intended vulnerabilities of the challenge programs. In cases where a Blue team believes that there is a vulnerability when none was intended, the EL team will re-evaluate the challenge application (using the justification provided by the team) to determine if an unintended vulnerability is

present in the challenge application. If there is an unintended vulnerability, the EL will determine there is an in-scope unintended vulnerability. If there is an in-scope unintended vulnerability, then the challenge program status will be updated to act as though that unintended vulnerability was intended, and all Blue teams' answers will be rescored against this updated classification. If there is an unintended out-of-scope vulnerability, then the status of the challenge program will not be updated (so other Blue teams' scores will not change) but the team that found the unintended out-of-scope vulnerability will be scored as if the challenge program in question is vulnerable. Note, this means that the denominators of the metrics for each Blue team will be different even under the second data segmentation.

4.7.1.5.2 Initial Assessments and Refinements Based on Blue Team Feedback

After the live collaborative engagement (2/8/2019), the EL team will provide an initial assessment to DARPA of the performance of the R&D Teams and the control team against the program metrics using the intended/confirmed vulnerabilities. The EL team will work with the AC teams and the Blue Teams to finalize the engagement results by 3/28/2019.

4.7.2 Engagement Results

4.7.2.1 Introduction

Engagement 7, the final engagement in the 4-year STAC program, took place from January 14th, 2019 to February 7th, 2019. This final Engagement was designed to measure the combined performance of the STAC research tool portfolio against that of the Control Team – a shift in focus from prior engagements. This result enabled us to draw direct conclusions about the advancements in the state-of-the-art achieved by the R&D performers, with the assumption that the performance of the Control Team represents the current state-of-the-art in Algorithmic Complexity (AC) and Side Channel (SC) vulnerabilities in Time and Space.

Engagement 1 – 4 focused on measuring the performance of the individual research approaches of each R&D performer compared to the control Team and distinguishing tools with unique capabilities from those that overlap. Engagements 5 and 6 measured the individual performance of each R&D performer while assessing the combined performance of the STAC tool portfolio through collaborative engagements. Engagement 7, by contrast, was designed to allow a direct comparison between the performances of the Control Team and R&D performers by equalizing the effort measured in total hours spent on the Engagement. The R&D performers were given one-week of offsite prep time, then a one-week collaborative engagement. The Control Team was given three weeks to work on the engagement offsite. With 5 R&D performers limited to 4 people per team during the live-collaborative engagement, this was equivalent to the approximate total effort of the Control Team.

The results of the engagement (Figure 80) show that the R&D performers outperformed the Control Team in three of the five vulnerability categories – AC Time (by 27 percentage-points) and Space (by 16 percentage-points), and SC Time (by 14 percentage-points). In the category of SC Time and Space questions, the difference of 2.6 percentage points between the Control Team and the R&D performers is assessed as approximately equivalent. In the category of SC Space vulnerabilities, the Control Team outperformed the R&D performers by 25 percentage-points. Under the assumption that the Control Team represents the state-of-the art in these vulnerability categories, we can conclude that the performance of the R&D performers in Engagement 7

relative to the Control Team indicate that they advanced the state-of-the-art in Algorithmic Complexity vulnerabilities in Time and Space, and in Side Channel vulnerabilities in Time.

The remainder of this report provides both a quantitative analysis that applies various metrics and a qualitative analysis of the Engagement 7 results and an assessment of the performance of the Adversarial Challenge (AC) Teams.

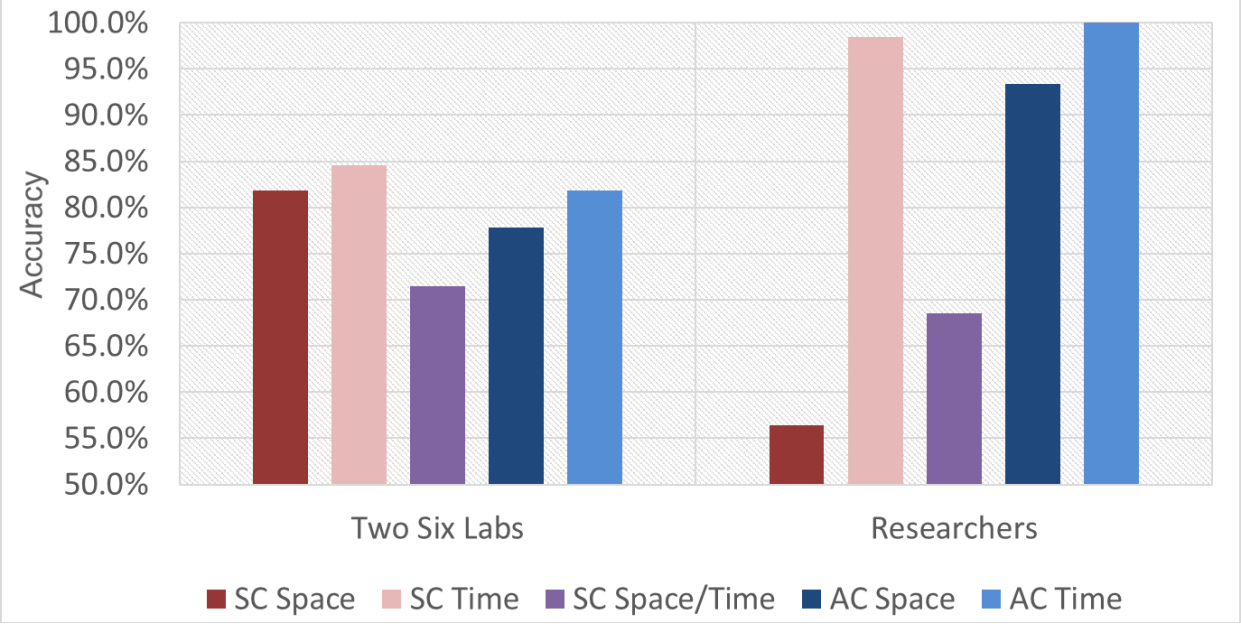


Figure 80: Engagement 7 accuracy by challenge question category, Control Team (Two Six Labs) versus R&D Performers

4.7.2.2 Quantitative Engagement Results

4.7.2.2.1 Overview

In Engagement 7 (E7), 60 questions were asked about 21 challenge programs. Of the 15 challenge programs, 12 were provided by CyberPoint and 9 were provided by Raytheon/BBN Technologies (BBN). The breakdown of challenge questions by question type (Algorithmic Complexity (AC) in Time, AC in Space, Side Channel (SC) in Time, SC in Space, and SC in Space and Time) and vulnerability category (Vulnerable and Non-Vulnerable) is shown in Table 31 below.

Table 41: How many questions were asked for a given question type and intended question result.

Question Type	Question Result	Number of Questions
Algorithmic Complexity in Time	Positive	5
Algorithmic Complexity in Space	Positive	7
Side Channel in Time	Positive	6
Side Channel in Space	Positive	5
Side Channel in Time and Space	Positive	3
Algorithmic Complexity in Time	Null	6
Algorithmic Complexity in Space	Null	11
Side Channel in Time	Null	7
Side Channel in Space	Null	6
Side Channel in Time and Space	Null	4

Five R&D performers and one Control Team, collectively known as the Blue Teams, participated in the engagement. All engagement participants answered all 60 challenge questions. Table 32 provides a listing of the teams that participated.

Table 42: The different teams that participated in Engagement 6

Blue Teams	Research Teams	University of Colorado Boulder (UC Boulder)
		GrammaTech
		Iowa State University (ISU)
		Northeastern University (NEU)
		Vanderbilt University
	Control Team	Two Six Labs
Adversarial Challenge (AC) Teams		CyberPoint
		BBN

** The University of Utah team did not participate in the live collaborative engagement and is not included in the analysis of the STAC tool portfolio. The team independently worked on the Engagement 7 challenges and their responses and results are included in the Engagement 7 appendix.

4.7.2.2.2 Performance Metrics

There are two sets of metrics used to evaluate the engagement performance. The first metric is the number of responses versus accuracy, which was more useful earlier in the program when performers were not required to answer every question. As the program goal is to develop tools to both identify and rule out vulnerabilities, we established a second set of metrics to distinguish the ability to identify versus rule out vulnerabilities. True Positive Rate (TPR) versus True Negative Rate (TNR) enable us to better understand the ability of performers to both identify and rule out vulnerabilities (We assume that vulnerable challenge programs contain intended vulnerabilities created by the AC Teams or unintended vulnerabilities discovered by the performers and that all other challenge programs are non-vulnerable).

- True Positive Rate (Number of correct “Yes” responses/ Total number of vulnerable challenge questions)
- True Negative Rate (Number of correct “No” responses/ Total number of non-vulnerable challenge questions)

Using the first metric of total accuracy versus number of responses, Figure 68, the R&D performers outperformed the Control Team (Two Six Labs) by 6 percentage points. To further understand the differences in performance between the R&D performers and the Control Team, other metrics must be applied.

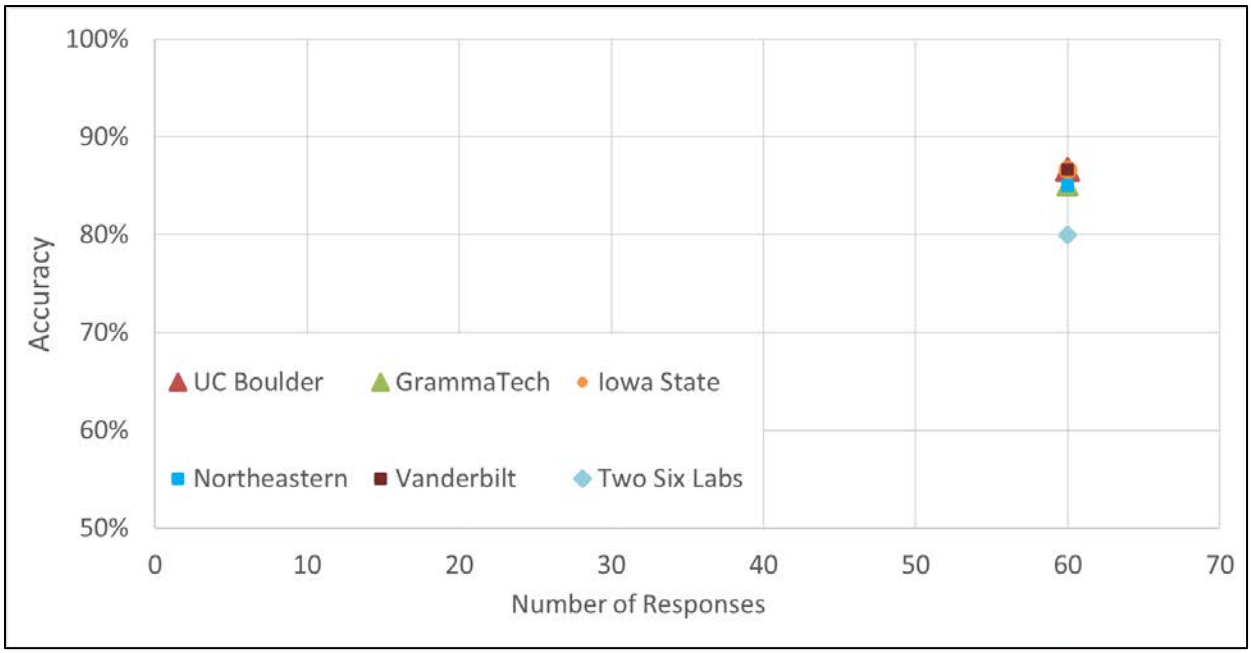


Figure 81: E7 reviewed accuracy versus number of responses. Accuracy only shown from 50% to indicate performance gain over 50-50 guess

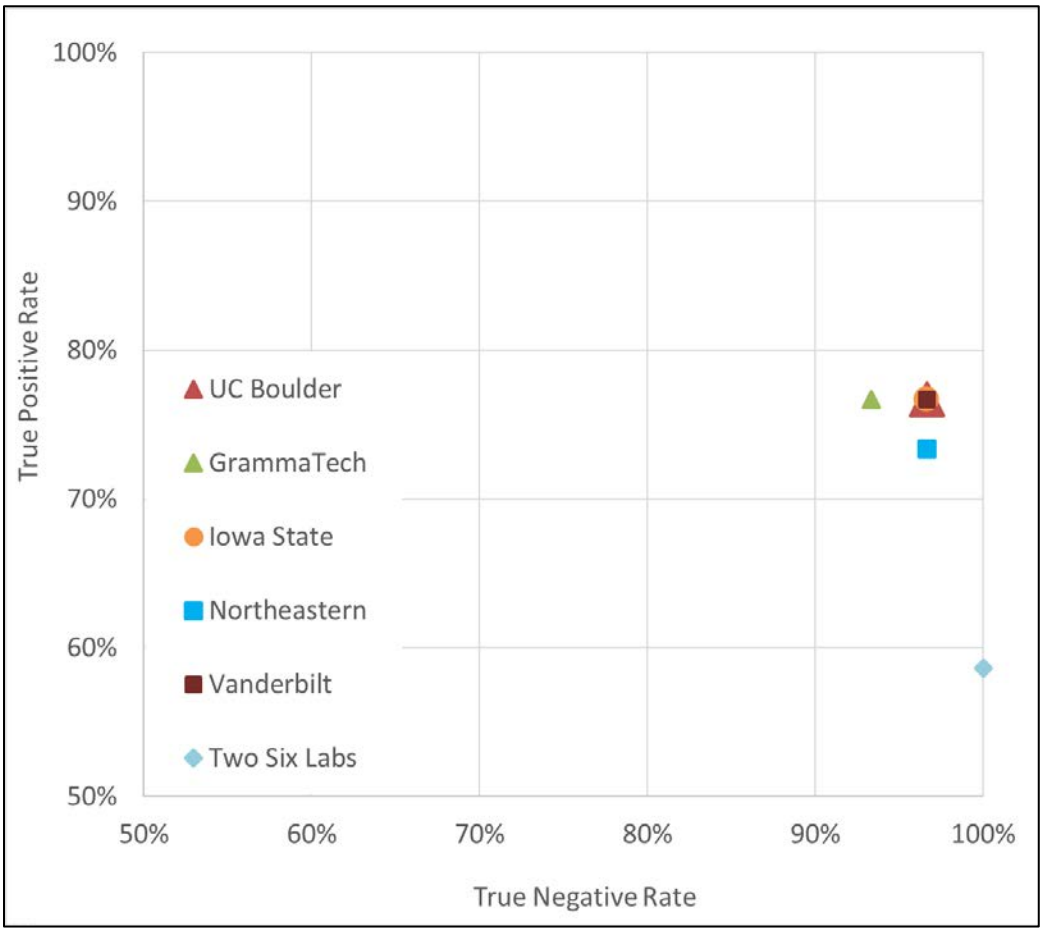


Figure 82: E7 TPR vs TNR – results only shown for top right quadrant of 50% to 100% for both TPR and TNR

TPR vs TNR, Figure 82, provides a better metric to distinguish the the performance of the R&D performers and the Control Team. R&D performers achieved a TPR of between 73% and 77% and a TNR of between 93% and 97%. The Control Team achieve a TPR of 58.6% and a TNR of 100%. These results show that the R&D performers outperformed the Control Team in TPR by approximately 17 percentage-points while only losing 4 percentage-points of TNR performance. When the control team ruled out an application as non-vulnerable it was non-vulnerable; however, relying only on current state-of-the-art tools, there were vulnerabilities that the Control Team could not find that the R&D performers identified. The R&D performers outperformed the current state-of-the-art in vulnerability detection on average with only a marginal loss in the ability to rule out vulnerabilities.

The metrics of total accuracy and TPR versus TNR fail to distinguish performance by question category. The results by question category, shown in Figure 80 in the Introduction and again in Figure 83, provide a clearer picture of the strengths of the STAC R&D tools relative to the current state-of-the-art. The STAC performers outperformed the Control Team in Algorithm Complexity and SC Time categories, approximately matched the Control Team performance in SC Time and Space, while failing to match the Control Team's SC Space performance.

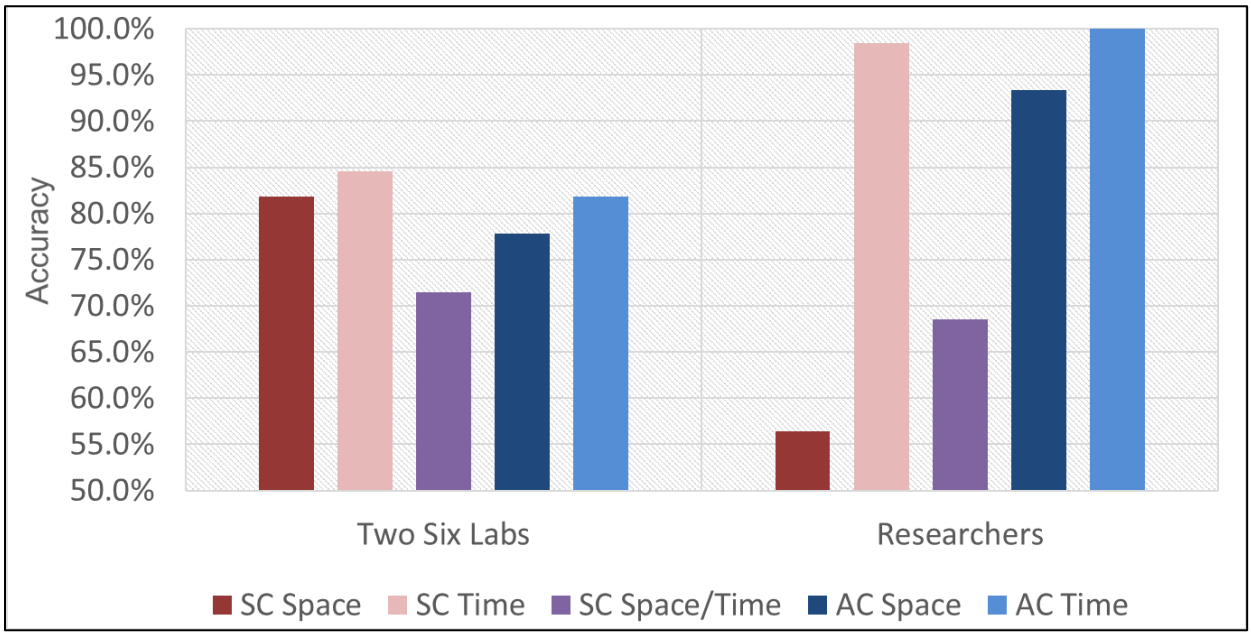


Figure 83: E7 accuracy by challenge question category, Control Team (Two Six Labs) versus R&D Performers

The next section takes a closer look at the categories of incorrect responses for the R&D performers compared to the Control Team to provide yet a clearer picture of the limitations of both the R&D performer tools and the current state-of-the-art tools used by the Control Team.

4.7.2.2.3 Categories of Incorrect Responses

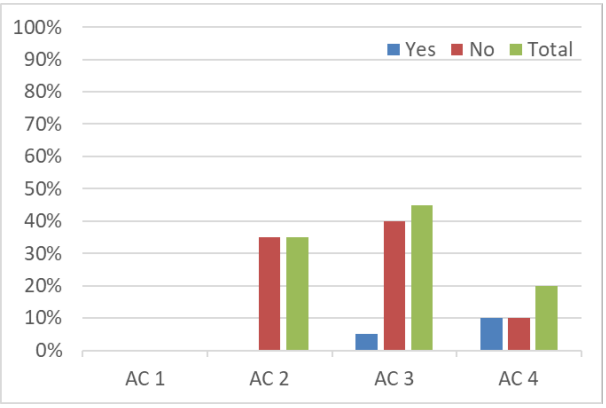
To provide a clearer picture of limitations of different performers, we established four categories of why performers answered a question incorrectly. These categories are not identical for AC and SC questions, but they mirror each other. For AC questions, the incorrect response categories are:

AC 13. Did not follow directions or violated the question definitions
AC 14. Failed to identify the complexity within the challenge program
AC 15. Failed to understand the input to the complexity control flow of the challenge program
AC 16. Failed to properly determine the complexity bounds of an identified complexity

For SC questions, the incorrect response categories are:

SC 13. Did not follow directions or violated the question definitions
SC 14. Failed to identify the secret or the secret location within the challenge program
SC 15. Failed to identify the side channel (processing conditioned on the secret value)
SC 16. Failed to properly analyze the side channel strength

An incorrect response can fit into multiple categories e.g. given a strong side channel a team identifies a weaker side channel. This is classified as both a failure to identify the intended side channel vulnerability and a failure to correctly determine strength.



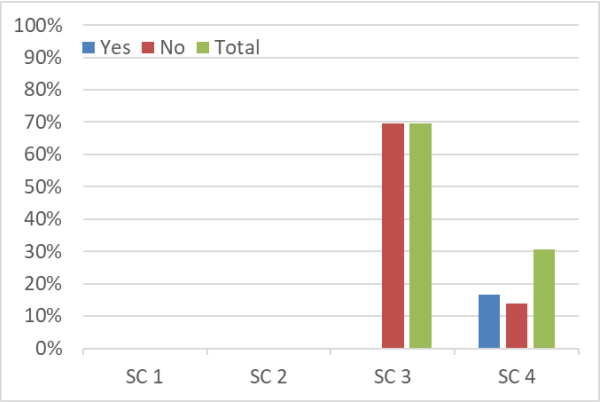
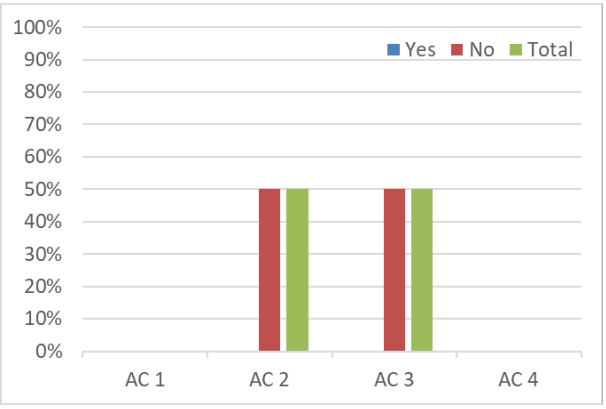


Figure 84: E7 R&D performers categories of incorrect responses: algorithmic complexity (left); side channel (right)



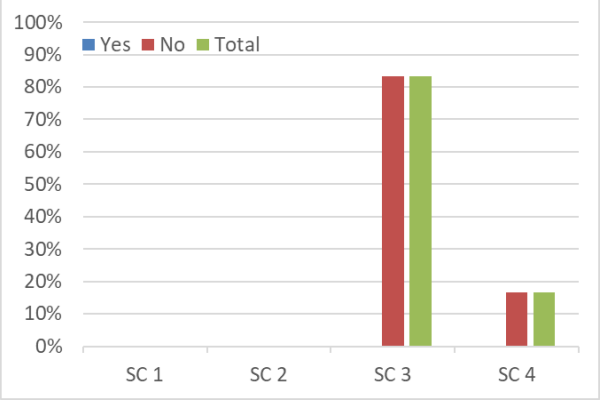


Figure 85: E7 Control Team categories of incorrect responses: algorithmic complexity (left); side channel (right)

Figure 84 shows the categories of incorrect responses for the R&D performers and Figure 85 shows the data for the Control Teams. The results show the categories of incorrect responses for the R&D performers and the Control Team were quite similar. They both struggled with identifying the complexities with the challenge program (AC 2) and understanding the input to the complexity control flow (AC 3). In these cases, both groups declared the challenges non-vulnerable a majority of the time. When teams identified the input to a complexity control flow (AC 3), usually both groups correctly evaluated the complexity bounds. This is reflected in the fact that for both performer groups, the AC 4 category of incorrect responses accounts for fewer than 20% of the incorrect responses.

The results for Side Channel questions show that for both performer groups, the overwhelming reason for incorrect responses (69% for R&D performers and 83% for the Control Team) was due to an inability to find the Side Channel within the challenge program (SC 3). In all cases of this category of incorrect, the performers incorrectly declared the challenges non-vulnerable as a result.

4.7.2.2.4 Blue Team Responses

Program	Question	Type	Vuln?	Northeastern	UC Boulder	GrammaTech	Iowa State	Vanderbilt	Two Six Labs
calculator_3	Question_023	AC Space	Y	Y	Y	Y	Y	Y	N
calculator_3	Question_025	AC Space	N	N	N	N	N	N	N
calculator_3	Question_034	AC Time	Y	Y	Y	Y	Y	Y	Y
calculator_4	Question_036	AC Space	Y	N	Y	Y	Y	Y	Y
calculator_4	Question_043	AC Time	N	N	N	N	N	N	N
calculator_4	Question_045	AC Space	Y	N	N	N	N	N	N
cyberwallet_1	Question_006	AC Space	N	N	N	N	N	N	N
cyberwallet_1	Question_035	SC Space	Y	Y	Y	Y	Y	Y	Y
cyberwallet_1	Question_051	SC Time	N	N	N	N	N	N	N
cyberwallet_2	Question_024	SC Space	N	N	N	N	N	N	N
cyberwallet_2	Question_059	SC Time	Y	Y	Y	Y	Y	Y	N
doormaster	Question_029	AC Time	N	N	N	N	N	N	N
doormaster	Question_041	SC Time	Y	Y	Y	Y	Y	Y	N
emu6502	Question_004	SC Time	N	N	N	N	N	N	N
emu6502	Question_012	SC Space	Y	Y	Y	Y	Y	Y	Y
emu6502	Question_014	SC Time	N	N	N	Y	N	N	N
emu6502	Question_016	AC Time	N	N	N	N	N	N	N
inandout_1	Question_007	AC Time	Y	Y	Y	Y	Y	Y	Y
inandout_1	Question_027	AC Space	Y	Y	Y	Y	Y	Y	Y
inandout_1	Question_049	SC Time	Y	Y	Y	Y	Y	Y	Y
inandout_1	Question_055	SC Space	N	N	N	N	N	N	N
inandout_2	Question_028	SC Time/Space	Y	N	N	N	N	N	N
inandout_2	Question_039	AC Time	Y	Y	Y	Y	Y	Y	Y
inandout_2	Question_053	AC Space	N	N	N	N	N	N	N
litmedia_1	Question_009	AC Space	Y	Y	Y	Y	Y	Y	Y
litmedia_1	Question_032	SC Time	N	N	N	N	N	N	N
litmedia_1	Question_056	SC Time/Space	N	N	N	N	N	N	N
litmedia_2	Question_001	SC Time	N	N	N	N	N	N	N
litmedia_2	Question_050	SC Time	N	N	N	Y	N	N	N
litmedia_2	Question_058	SC Time/Space	N	N	N	Y	N	N	N
phonemaster	Question_003	SC Time	Y	Y	Y	Y	Y	Y	Y
power_state	Question_031	AC Space	N	Y	Y	Y	Y	Y	N
power_state	Question_037	AC Time	Y	Y	Y	Y	Y	Y	Y
psa	Question_010	AC Time	Y	Y	Y	Y	Y	Y	N
psa	Question_030	AC Space	Y	Y	Y	Y	Y	Y	N
roulette_1	Question_013	SC Time/Space	N	N	N	N	N	N	N
roulette_1	Question_019	AC Space	N	N	N	N	N	N	N
roulette_1	Question_048	SC Space	N	N	N	N	N	N	N
roulette_1	Question_054	AC Time	N	N	N	N	N	N	N
roulette_2	Question_011	SC Time/Space	Y	Y	Y	Y	Y	Y	Y
roulette_2	Question_033	SC Space	Y	N	N	N	N	N	N
roulette_2	Question_046	AC Space	N	N	N	N	N	N	N
securgate	Question_008	SC Space	Y	Y	Y	Y	Y	Y	Y
securgate	Question_018	AC Time	N	N	N	N	N	N	N
suspicion_1	Question_020	AC Space	N	N	N	N	N	N	N
suspicion_1	Question_026	SC Space	Y	N	N	N	N	N	Y
suspicion_1	Question_040	SC Time/Space	Y	N	N	N	N	N	N
suspicion_2	Question_022	SC Time/Space	N	N	N	N	N	N	N
suspicion_2	Question_044	AC Space	Y	Y	Y	Y	Y	Y	Y
suspicion_2	Question_057	SC Space	N	Y	Y	N	Y	Y	N
suspicion_2	Question_060	AC Space	Y	Y	Y	Y	Y	Y	Y
thermomaster	Question_021	AC Space	N	N	N	N	N	N	N
thermomaster	Question_038	SC Time	N	N	N	N	N	N	N
thermomaster	Question_052	AC Time	Y	Y	Y	Y	Y	Y	N
wordshare	Question_002	SC Space	Y	N	N	N	N	N	Y
wordshare	Question_005	AC Space	N	N	N	N	N	N	N
wordshare	Question_047	SC Time	N	N	N	N	N	N	N
yapmaster	Question_015	AC Space	Y	Y	Y	Y	Y	Y	N
yapmaster	Question_017	SC Time	N	N	N	N	N	N	N
yapmaster	Question_042	SC Space	Y	N	N	N	N	N	N

	Researchers	Control Team
1	✓	×
2	×	✓
3	×	×

Figure 86:Blue Teams Responses (green - correct, red - incorrect) with legend for three categories of interesting questions. E.g. category 2, highlighted in blue indicates where researchers got a question correct that the control team missed.

4.7.2.3 Qualitative Analysis

The engagement responses for the R&D performers and the Control Team (Figure 86) allow us to further investigate the performance differences between the two performer groups. During the engagement, the five R&D performers were required to collaborate but not required to reach consensus on their responses. For all 60 challenge questions, either four or five R&D performers reached a consensus. In the analysis below we use the consensus response to assess the performance of the R&D performers relative to that of the Control Team.

We identified three distinct response patterns between the R&D performers and the Control team that allow us to better understand the strengths of the STAC R&D performers. These three response patterns are mapped to into three categories of challenge questions.

Category 1. R&D performers successfully answered but the Control Team failed to successfully answer (highlighted in blue in Figure 86). These challenge questions contain the underlying properties that illustrate the advancement in the state-of-the-art by R&D performers.

Category 2. R&D performers failed to answer successfully but the Control Team answered successfully (highlighted in violet in Figure 86). We believe that these challenge questions/programs represent areas that the R&D performers did not focus on advancing the state-of-the-art.

Category 3. Neither the R&D performers nor the Control Team answered successfully. Challenge programs in this category contained underlying properties beyond the current analysis capabilities of both the STAC R&D tools and the current state-of-the-art tools.

Section 4.7.2.3.1 discusses the Category 1 challenge questions. Category 1, where the R&D performers identified vulnerabilities missed by the Control Team, contained AC Time, AC Space, and SC Time questions, but no SC Space or SC Time/Space questions. This supports the hypothesis that the R&D performers advanced the state-of-the-art in the Algorithmic Complexity (AC T and AC S) and SC Time domains but not in the SC Space (and by extension SC Time/Space) domain.

Despite the achievements of the STAC R&D performers, their tools have difficulty analyzing complex encryption schemes, and performing automated analysis over a sufficiently large complex code base (where the threshold for sufficiently large varies by the tool and the heuristic filters used to identify the potentially vulnerable code). A human analyst is required to use context clues to identify an initial coarse region of code and to annotate the application before the semi-automated analysis tools can be effective. If the region of code to be analyzed is too broad, or the interactions are sufficiently complex, the R&D performers are unable to find the vulnerability. Section 4.7.2.3.2 and Section 4.7.2.3.3 discuss respectively the Category 2 and Category 3 challenge questions.

4.7.2.3.1 Category 1: Distinguish STAC R&D Performer Capabilities

Category 1 contains the eight challenge questions that the R&D performers successfully answered, and the Control team failed to answer corrects. The R&D performers either identified the intended vulnerability or an unintended vulnerability.

4.7.2.3.1.1 Calculator 3 Question 023 (AC Space, Unintended Vulnerable)

4.7.2.3.1.1.1 Unintended Vulnerability Discovered by R&D Performers

The R&D performers developed attacks that exploit the inefficiency of subtraction and division code which have long-running loops with long arrays. These loops incrementally append coefficients to arrays for an inefficient use of memory. The R&D performers were also able to successfully use this vulnerability for the other variant of the Calculator application, Calculator 4 Q036. Some R&D performers identified the potentially vulnerable code with dynamic analysis, but given their limited offline analysis time, they relied on the exploits manually generated by other R&D performers to confirm the potentially vulnerable code as vulnerable.

4.7.2.3.1.1.2 Control Team

The Control Team identified no mechanisms in pre-processing or calculator operations to exceed the memory resource usage limit.

4.7.2.3.1.2 Cyberwallet 2 Question 059 (SC Time, Intended Vulnerable)

4.7.2.3.1.2.1 Intended Vulnerability Discovered by R&D Performers

The CyberWallet challenge program is a banking application. This variant contained an intended SC Time vulnerability that allows an attacker to transfer funds from another user's bank account into their own. This required the analysts to link two vulnerabilities together: a side channel that enables an attacker to leak another user's secret account number; and a "broken" input guard that allows the transfer of money without authenticating the transferrer.

Account numbers are stored in a trie data structure with each digit of the account number corresponding to a branch in the trie. When a transfer is initiated, to a target account, each the application performs a digit by digit comparison to match the input to an existing account. This introduces an observable timing side channel that leaks an existing account number digit-by-digit.

The R&D performers successfully identified the leaked account numbers and the mechanism to use the leaked account number to take money from another user's account.

4.7.2.3.1.2.2 Control Team

The Control Team identified the mechanism to initiate a transfer from a target user's account into the attacker's own account, but did not discover the side channel to leak the target user's secret account number required for the exploit to succeed.

4.7.2.3.1.3 DoorMaster Question 041 (SC Time, Intended Vulnerable)

4.7.2.3.1.3.1 Intended Vulnerability Discovered by R&D Performers

The DoorMaster application is a physical access control system with a door and a backing endpoint. The door validates presented key fobs and allows a user to create a new key fob, with each keyfob tied to a public private key pair. The door application requires a constant connection with the backing endpoint, if this connection is broken, the application locks out all users.

The application contains an intended SC Time vulnerability that leaks the private keys of existing key fobs, enabling an attacker to gain access with another user's key. The R&D performers successfully discovered that the frequency of the ping used to verify the connection with the backing endpoint leaks the private keys of all registered key fobs bit-by-bit.

4.7.2.3.1.3.2 Control Team

The Control Team observed the public-private key pair generation process and the authentication process for validating a key fob but saw no information leak in time. They noted that critical operations used fast HashMap lookups and the hand-rolled PRNG uses constant value. They fully exhausted the small codebase via manual analysis. They observed that on each keygen, the PRNG seed is set to previously generated key, but determined that since it uses SecureRandom, there was no information leakage.

4.7.2.3.1.4 PSA Question 010 (AC Time, Intended Vulnerable)

4.7.2.3.1.4.1 Intended Vulnerability

The application is a Python Static Analyzer that accepts inputs of python bytecode (pyc) and allows users to perform various static analysis tasks. The application uses contextualized RDA (sic) algorithms to convert program instructions into graph format. This implementation can result in combinatorial complexity – An instruction can have multiple possible impacts depending on program state. This implementation RDA (sic) simulates enough of the state then computes each instruction's impact on that state. Instructions that fork multiple states multiply the contexts that all future instructions need to deal with resulting in a combinatorial explosion of contexts PSA's allows the user to configure the maximum depth of the virtual call stack that they want to maintain. Setting this to a high number (20+) and attempting to analyze a heavily recursive program will cause an explosion in execution contexts. The analysis will take a long time to terminate, blocking all future queries.

4.7.2.3.1.4.2 Unintended Vulnerability Discovered by R&D Performers

The R&D performers discovered that submitting a POST request for an analysis that doesn't exist creates a lock file. Subsequent GET request submissions for that nonexistent analysis causes a thread to be allocated for each request, with each thread blocking indefinitely on the lock file. Submitting 200 of these requests exhausts the application's thread pool causing it to fail to process future requests.

4.7.2.3.1.4.3 Control Team

The control team analyzed recursive and loop structures in the application and concluded that the application was nonvulnerable.

4.7.2.3.1.5 PSA Question 030 (AC Space, Intended Vulnerable)

4.7.2.3.1.5.1 Intended Vulnerability

An attacker can combine PSA's write to disk functionality with a verbose logging mode to exceed the resource usage limit. An optional "verbose-on-error" mode in the serializer module serializes every revision of every module if the first pass of PSA produced an error. Combining this option with "retry-on-error" and a virtual call stack of at least 40 will trigger this vulnerability.

4.7.2.3.1.5.2 Unintended Vulnerability Discovered by R&D Performers

The Researchers found that pretty-printing a JSON file adds spacing characters and PSA does not check the validity of a JSON file before pretty-printing to disk. They were able to create an exploit demonstrating this unintended vulnerability that exceeds the resource usage limit.

4.7.2.3.1.5.3 Control Team

The Control Team declared the challenge nonvulnerable, asserting that PSA doesn't store any considerable volume or quantity of files and the input budget is too small to request many small files. Zip bombs were investigated for potential vulnerabilities, but all identified pathways were caught by PSA's safeguards.

4.7.2.3.1.6 ThermoMaster Question 052 (AC Time, Intended Vulnerable)

4.7.2.3.1.6.1 Intended Vulnerability

The application is a networked system for power plant temperature control and prediction. ThermoMaster supports multiple thermostat users who send target setpoint and current temperature readings to the server. The server responds with adjustments to reach the target setpoint.

The application's intended vulnerability relies on computations with subnormal floating-point values. Subnormal (close to zero) floating point computations take longer than other floating-point computations. An attacker can bypass an input guard intended to prevent these subnormal values and force computation on subnormal floating-point numbers to delay another user's request.

4.7.2.3.1.6.2 Unintended Vulnerability Discovered by R&D Performers

The Researchers found unintended vulnerabilities that caused sufficient delays. The R&D performers found a range of inputs with slow computation time without having to bypass the guard that prevents subnormal floating points. Their proofs of vulnerabilities disabled temperature control measures and set the current temperature to –FloatMinValue, close to 0, or other edge values. Some of the performers' tools reported the suspicious loops that were then used to generate these proofs of vulnerabilities.

4.7.2.3.1.6.3 Control Team

The Control Team found no code outside the PID that could have an AC time effect. The login uses a fixed number of iterations and some set constant commands are non-functional. They did note that changing the temperature metric from C to F can allow the user to exceed the intended max temperature, but determined that it had no effect on time.

4.7.2.3.1.7 YapMaster Question 015 (AC Space, Intended Vulnerable)

4.7.2.3.1.7.1 Intended Vulnerability Discovered by R&D Performers

YapMaster is a secure message application that enables users to share private messages – Yaps. Each Yap is encrypted with a unique key. YapMaster allows a user to submit a GET request for a Yap that does not yet exist and has a very large ID value. The Yap Server then bypasses the cache and attempts to get the Yap message from the Random-Access Memory Mapped disk store. The Memory Mapped disk store then multiplies the index of the requested non-existent Yap by the normal Yap message size and expands the size of the memory mapped buffer by this size.

4.7.2.3.1.7.2 Control Team

The Control Team incorrectly determined that the only interface to write to disk file was through a writer object. They saw that the ID affects where in the file the message is written and that a large ID will write to a large offset. The Control Team failed to find a path to bypass the input

guard to set the large ID value, while the R&D performers found the intended path to bypass the input guard.

4.7.2.3.2 Category 2: STAC R&D Performer Limitations

4.7.2.3.2.1 Suspicion 1 Question 026 (SC Space, Intended Vulnerable)

4.7.2.3.2.1.1 Intended Vulnerability Identified by Control Team

Suspicion is a multi-player peer-to-peer game with one player randomly assigned as the spy and the remaining players are informed of a secret location and secret password. The goal of the game is for the spy to determine the secret location and for the other players to determine the identity of the spy. Users send questions and answers to one another to try to win the game.

The secret location is leaked via the sizes of the first three answer messages sent by the game leader. There are a fixed set of 26 possible locations stored in a list. Each of the first three answer messages leaks one trit of the base 3 represented index of the secret location. There exists an explicit mapping from the observed message sizes to the location index. Given the first three message sizes as s_1, s_2, and s_3, the location index, i, is $i = 9 * (s_1 - 1002) + 3 * (s_2 - 1002) + s_3 - 1002 = 9s_1 + 3s_2 + s_3 - 13026$.

The Control Team accurately identified this intended vulnerability and observed that the provided client can be modified to observe the message sizes and leak the secret location.

4.7.2.3.2.1.2 R&D Performers

The R&D performers found parts of the intended vulnerability, including the possible 3-digit leak of the location. They only looked at passive observations of packet traffic and observed that the packets over TCPDUMP all appeared to be consistently padded. They failed to identify that while the passive observations of message traffic was insufficient to leak the secret, the client code could be modified to leak the unique sizes of the first three messages received by all parties.

4.7.2.3.2.2 Wordshare Question 002 (SC Space, Unintended Vulnerable)

4.7.2.3.2.2.1 Unintended Vulnerability Discovered by Control Team

Wordshare is a web-based word processing application that enables users to securely store documents. Privileged users can encrypt secret words in documents. These words appear as cyphertext to non-privileged users.

The Control Team found an unintended vulnerability in the fact that non-privileged users can view and add secure docs. An attacker with non-privileged credentials can get the target document with the secret encrypted word and resubmit same secret word multiple times with different initialization vectors (IVs). If bytes of decrypt fall outside a range, the application tries to fill in value from dictionary. If the value is a space, the number of bytes written is reduced. The buffer has a max size and attacker can pre-fill buffer near the limit. To exploit, pick the value that if a space is not in decrypt, the buffer is filled. Check if length of response is null or ok. Use dictionary with space chars in first or second position, chose IV, and can determine if a byte of decrypt is in the given range. Binary search each byte to identify specific bytes in plaintext secret.

4.7.2.3.2.2.2 R&D Performers

The R&D Performers claimed that there did not exists a sufficient mapping between the secret encrypted word and public data available to non-privileged users. They did observe slight differences in packet sizes different modes of the applications. Additionally, they observed a way to potentially leak the secret, but the pathway they found was filtered out by the server. This example illustrates the R&D performers' difficulties with analyzing both encryption and sufficiently complex input pathways required to trigger a vulnerability.

4.7.2.3.3 Category 3: Limitations of Both STAC R&D Performers and Current State-of-the-Art

4.7.2.3.3.1 Calculator 4 Question 045 (AC Space, Intended Vulnerable)

4.7.2.3.3.1.1 Intended Vulnerability

A "bug" allows posts that slightly exceed the guarded maximum post size, and another expands the actual contents of a post to many times its actual size. The input stream is compared byte by byte with a "random" stream (with fixed seed) and if that byte value (v) is the same, the next v reads will return v. If there is no overlap between the post content and the sequence of random characters, the post will not be altered at all. The more overlap there is, the more the post content will be expanded. An attacker can cause the post content to expand by entering a particular sequence of upper-case characters in the calculator expression box.

4.7.2.3.3.1.2 R&D Performers and Control Team

Both the R&D performers and the Control Team focused on trying to write to the Logfile via a StackOverflow or Execution exceptions but were unable to write more than a few kB to it. No team discovered the explicit link between the user input and the expansion of content written to disk.

4.7.2.3.3.2 InAndOut 2 Question 028 (SC Time/Space, Intended Vulnerable)

4.7.2.3.3.2.1 Intended Vulnerability

InAndOut is an online order-placing and tracking application for a pizza parlor. A secret confirmation code is used to pickup placed orders and an intended vulnerability leaks this confirmation code. The confirmation code is a string of the form #-#-#-#-#-# calculated from a random number that is translated to the above form via binary expansion. A different Linear Feedback Shift Register (LFSR) is used for each field with the computation time dependent on the length of the LFSR. An attacker can build a mapping of response times to possible confirmation code strings and map an observed response time to a confirmation code. Each response is delayed by a random offset which is coded in the size of a particular message from the inventory manager. This size must be mapped to the offset value to enable an observed response time (minus the offset value) to be mapped to a particular confirmation code string.

4.7.2.3.3.2.2 R&D Performers' Response

The R&D performers noted a correlation in the sizes of different inputs but only a weak and noisy SC in time. They observed that packets contain toppings and random values and that the validation code does not relate to the size of the packets. They failed to identify the SC space vulnerability that enables an attacker to calculate the random time offset to then enable mapping response times to secret confirmation codes.

4.7.2.3.3.2.3 Control Team

The Control Team identified the distribution of the random offset, and similar to the R&D performers failed to map this to the space component of the side channel.

4.7.2.3.3.3 Roulette 2 Question 033 (SC Space, Intended Vulnerable)

4.7.2.3.3.3.1 Intended Vulnerability

Roulette is an application that allows players to play virtual games of roulette. An intended Side Channel allows a third party to decrypt application messages. The application's network traffic is encrypted using CBC encryption, without a MAC, which is vulnerable to a "Padding Oracle Attack". Encryption can be broken by an attacker determining if a message is correctly padded, as each byte in the padding is the number of bytes of padding. If the application tries to decrypt a message with incorrect padding, it sends an error response. Otherwise, the decrypted message is passed on for handling but will fail because the contents don't represent a full message. An attacker, M, intercepts a message between benign users A and B. M repeatedly resends the packet to B, updating it to appear like it came from A. M tries all possible values for the last byte, noting which does not cause a padding-error response. M is able to undo the encryption of all but the first block of the message.

4.7.2.3.3.3.2 R&D Performers

The R&D performers observed that the sizes of the bet packets are related to the content but concluded that there wasn't not enough information to reveal the content from the sizes alone. Potential exploits requiring additional credentials or a weaker threshold for exploitability were proposed, but none of the R&D performers was able to find the intended vulnerability.

4.7.2.3.3.3.3 Control Team

The Control Team observed that the bet packet is of a consistent format and concluded that there are too many bytes of information to determine the secret.

4.7.2.3.3.4 Suspicion 1 Question 040 (SC Time/Space, Intended Vulnerable)

4.7.2.3.3.4.1 Intended Vulnerability

Suspicion 1 applies a variant of the Padding Oracle Attack side channel from Roulette 2 to leak the secret. In this instance the attacker must identify and decrypt a specific packet, with the specific packet identified via a space side channel.

4.7.2.3.3.4.2 R&D Performers

The Researchers' tools identified the bounds on the secret password and confirmed from taint analysis that the password authentication functions do not leak the secret. One tool analyzed potential areas that could allow a segmented side channel and concluded that they were nonvulnerable because the algorithm iterates over all characters of submitted input even after the correct password has been identified. Another tool gave conflicting information that no loops were dependent on the password. Dynamic analysis showed that the size of the packets leaks the password length. None of these indicated a potential vulnerability of sufficient strength.

4.7.2.3.3.4.3 Control Team

The Control Team also noted the password is only checked in one method and that while the method performs a character-by-character chess of the password, the method doesn't terminate once an incorrect character is identified. Similar to the Researchers, they discovered that the

length of the password can be found, but as it is any combination of characters up to 5, it cannot be brute forced within the provided budget.

4.7.2.3.3.5 YapMaster Question 042 (SC Space, Intended Vulnerable)

4.7.2.3.3.5.1 Intended Vulnerability

The data structure used to store each user's Yaps, enables an attacker to write into data space intended for another user. A cache stores each user's Yaps in the order in which they are submitted and a hashmap is used to map a user's username to the user's cache. An attacker can trigger a string match collision with another user's username and cause the application to write to the other user's cache. This can be combined with a broken input guard on the length of a Yap to leak another user's Yaps one character at a time. An attacker can misrepresent the length of a Yap submission and cause the application to process on another user's Yaps. The application will then produce an observable byte that leaks a character of the other user's Yaps. The attacker can craft input messages to leak all the characters of another user's Yaps.

4.7.2.3.3.5.2 R&D Performers

The Researchers found that network traffic reveals the user, alias, and ID of the most recent yap plus encrypted traffic. Static analysis tools across different teams were used to approximate loop complexity and reason over different branch execution times, but no Side Channel was found. A team noticed that they can overwrite the most recent yap using user:aliasas alias and posting ID times yaps, but determined this was probably unintended; they think they need to post their messages before the target does which contradicts their understanding of the question description. If it was intended, another team stated there's no way to read contents of cache and network capture does not reveal time or space difference. Additionally, overwriting the cache does not appear to reveal the benign user's privatekey.

4.7.2.3.3.5.3 Control Team

The Control Team found after identifying the most recent yap, they can decrypt the message by brute forcing the key or attempting to falsely share the message to our YapMasteruser. They saw no mechanism to see if a yap was decrypted successfully or reduce the space of keys to more tractable number. They determined they can only get plain text message offline by brute forcing the key.

4.7.2.4 Adversarial Challenge (AC) Teams

The STAC AC Teams provided 21 challenge programs resulting in 60 challenge questions. The performance on challenge questions provided by the two AC Teams further distinguished the R&D performers from the Control Team. The R&D performers and the Control Team achieve approximately the same accuracy on CyberPoint challenge questions, with an 83% accuracy for the R&D performers and an 84% accuracy for the Control Team. However, on BBN challenge questions, while the R&D performers maintained approximately the same accuracy with 86%, the Control Team's accuracy was 68%.

4.7.2.4.1 CyberPoint

CyberPoint provided 12 challenge programs resulting in 38 challenge questions with 14 intended vulnerable and 24 intended non-vulnerable. Blue teams discovered unintended vulnerabilities that impacted 4 intended nonvulnerable CyberPoint challenge questions. The resulting distribution was 18 vulnerable and 20 non-vulnerable questions. CyberPoint applied a similar

variant generation approach used in the previous STAC engagements. Each challenge program has a benign functionality, but different variants use different algorithms to achieve the same benign functionality. Earlier in the program, the team used a greater number of variants and often, one unintended vulnerability in one variant would be present in all variants. Additionally, performers would look for patterns in the variants to help conclude whether a challenge question was likely to be vulnerable or nonvulnerable in cases where they did not have explicit PoVs. In later engagements, in particular in Engagement 7, the team took additional effort to reduce the number of variants of each challenge program to two, and to use sufficiently different algorithms in both the vulnerable and non-vulnerable variants. For both variants may be non-vulnerable and use different algorithms to achieve the same program behavior. This helps to discourage performers from assuming that a difference in variants is indicative of potential vulnerability.

4.7.2.4.1.1 Engagement Results

Table 43: R&D performer accuracy on CyberPoint challenges.

Challenge Program	Positive Qs	Null Qs	Total Qs	Attempts	Correct Attempts	Percentage Correct
Calculator_3	2	1	3	15	15	100%
Calculator_4	2	1	3	15	9	60%
Cyberwallet_1	1	2	3	15	15	100%
Cyberwallet_2	1	1	2	10	10	100%
InandOut_1	3	1	4	20	20	100%
InandOut_2	2	1	3	15	10	67%
Litmedia_1	1	2	3	15	15	100%
Litmedia_2	0	3	3	15	13	87%
Roulette_1	0	4	4	20	20	100%
Roulette_2	2	1	3	15	10	67%
Suspicion_1	2	1	3	15	5	33%
Suspicion_2	2	2	4	20	16	80%
Total	**18**	**20**	**38**	**190**	**158**	**83%**

Table 44: Control Team accuracy on CyberPoint challenges.

Challenge Program	Positive Qs	Null Qs	Total Qs	Attempts	Correct Attempts	Percentage Correct
Calculator_3	2	1	3	3	2	67%
Calculator_4	2	1	3	3	2	67%
Cyberwallet_1	1	2	3	3	3	100%
Cyberwallet_2	1	1	2	2	1	50%
InandOut_1	3	1	4	4	4	100%
InandOut_2	2	1	3	3	2	67%
Litmedia_1	1	2	3	3	3	100%
Litmedia_2	0	3	3	3	3	100%
Roulette_1	0	4	4	4	4	100%
Roulette_2	2	1	3	3	2	67%
Suspicion_1	2	1	3	3	2	67%
Suspicion_2	2	2	4	4	4	100%
Total	**18**	**20**	**38**	**38**	**32**	**84%**

4.7.2.4.1.2 Unintended Vulnerabilities

The R&D performers identified some unintended vulnerabilities dURIng the Engagement that impacted four intended non-vulnerable questions.

4.7.2.4.1.2.1 Calculator_3, Q023, AC Space (Memory)

The Researchers manually developed attacks that exploit the inefficiency of subtraction and division code which have long-running loops with long arrays. These loops incrementally append coefficients to arrays for an inefficient use of memory. They were able to successfully use this vulnerability for the other variant, Calculator_4 Q036, as well. Some teams located this potential vulnerable code with dynamic analysis but their time to analyze it was limited and relied on the exploits shared by other teams in the live collaborative portion.

4.7.2.4.1.2.2 InAndOut_1 Q007, AC Time

There was an intended 1-1 mapping between validation code and binary code. There is an error in the verification of a validation code which allows an attacker to permanently delay subsequent order requests. An attacker can order a pizza and then submit a pickup order using the resulting validation code. Instead of using the exact validation code, the attacker should change any non-zero digit in the validation code to a different digit. This will prevent any subsequent pickup requests from being processed. This can be used to make it look as though order is being processed when no such order is being processed.

All 7 teams identified an unintended vulnerability where the attacker submits an incorrect confirmation number whose binary code corresponds to a correct, in-use confirmation number. A Researcher's tool reported the method that contains the suspicious while loop that led to the exploit. During the live-collaboration, the Research Teams mentioned that this vulnerability was likely unintended.

4.7.2.4.1.2.3 Litmedia_1 Q009, AC Space (Memory)

The Researchers were able to prove that there was an algorithmic complexity in space in the IconGuide class. When a user requests an HTML page, the HTML page often contains a link to an image. The client has to send a HTTP GET request to obtain that image. All these operations are expensive in terms of memory usage. The attacker can send multiple HTTP GET requests to download images from the server, in a short time interval. This will create many parallel executions of the above code, each of them allocating memory for the images. In total, this will exceed the budget of 600 MB.

4.7.2.4.1.2.4 Suspicion_2 Q060, AC Space (Memory)

The Researchers were able to find a vulnerability in the method expand: with some specific values returned by the input data stream, it is possible to create a very large file as well as a very large memory usage: the result of the decoded message is first stored in a temporary file tmpFile, and then the content of this file is read into a byte array byte[] encodedBytes. The encoded message consist in a sequence of pairs (character, number of occurences), that tells what's the next character in the decoded message and how many times it occurs contiguously. For instance, the encoded sequence 'h',1,'e',1,'l',2,'o',1 is decoded into the message hello. Therefore, the encoded message is 'c',10000, the decoded message will be of length 10000. In the method expand, there is a conditional – *if ((aggregateContent += num) > 1000000)* - that ensures that the message is no longer than 1,000,000. However, because *num* can be negative, messages can be encoded as to not trigger the exception "Data exceeded maximum allowed size" – by using a negative value for the first number. An example of such input is 'c', -2000000000, 'c', 2000000000. This input will generate a file of 2Gb and then a memory usage of 2Gb.

4.7.2.4.2 Raytheon/BBN Technologies

BBN provided 9 challenge programs resulting in 22 challenge questions with 12 intended vulnerable and 10 intended non-vulnerable. Blue teams discovered unintended vulnerabilities that impacted 2 intended non-vulnerable BBN challenge questions. The resulting distribution was 12 vulnerable and 10 non-vulnerable questions, as 3 intended vulnerable questions were deemed unintentionally non-vulnerable.

4.7.2.4.2.1 Engagement Results

Table 45: R&D performer accuracy on BBN challenges.

Challenge Program	Positive Qs	Null Qs	Total Qs	Attempts	Correct Attempts	Percentage Correct
Doormaster	1	1	2	10	10	100%
Emu6502	1	3	4	20	20	100%
Phonemaster	1	0	1	5	5	100%
Power_state	1	1	2	10	10	100%
PSA	2	0	2	10	10	100%
Securgate	1	1	2	10	10	100%
Thermomaster	1	2	3	15	15	100%
Wordshare	2	1	3	15	10	67%
Yapmaster	2	1	3	15	10	67%
Total	**12**	**10**	**22**	**110**	**100**	**91%**

Table 46: Control Team accuracy on BBN challenges.

Challenge Program	Positive Qs	Null Qs	Total Qs	Attempts	Correct Attempts	Percentage Correct
Doormaster	1	1	2	2	1	50%
Emu6502	1	3	4	4	4	100%
Phonemaster	1	0	1	1	1	100%
Power_state	1	1	2	2	2	100%
PSA	2	0	2	2	0	0%
Securgate	1	1	2	2	2	100%
Thermomaster	1	2	3	3	2	67%
Wordshare	2	1	3	3	3	100%
Yapmaster	2	1	3	3	1	33%
Total	**12**	**10**	**22**	**22**	**16**	**72%**

4.7.2.4.2.2 Unintended Vulnerabilities

The Blue Teams identified some unintended vulnerabilities during the Engagement that impacted two intended nonvulnerable questions.

4.7.2.4.2.2.1 Emu6502 Q012, SC Space

The teams discovered that the initial values that seemed user-editable were discarded at the emulator's runtime, meaning each execution of an out#.s file followed the same instruction pattern and resulted in deterministic register value updates each run. They acknowledged that there could be some collisions in the set of 500 files and they only have 25 operations. The teams were able to use their tools to profile each program as it executed, finding that each program had a unique signature in the first 25 instructions that would allow them to guess which program was being executed with oracle queries.

4.7.2.4.2.2.2 Wordshare Q002, SC Space

This program was intended non-vulnerable, the Control Team found an Unintended Vulnerability that none of the Researchers noticed. A non-privileged user can view and add secure docs. They can get the target doc and resubmit same secure word multiple times with different IVs. If bytes of decrypt fall outside a range, application tries to fill in value from dictionary. If the value is a space, number of bytes written is reduced. The buffer has a max size and attacker can pre-fill buffer near the limit. To exploit, pick the value that if a space is not in decrypt, the buffer is filled. Check if length of response is null or ok. Use dictionary with space chars in first or second position, chose IV, and can determine if a byte of decrypt is in the given range. Binary search each byte to identify specific bytes in plaintext secret.

4.7.2.4.2.3 Unintended Non-Vulnerable

The Blue Teams identified some issues during the Engagement that impacted the following 2 intended vulnerable questions.

4.7.2.4.2.3.1 Emu6502 Q014, SC Time

4.7.2.4.2.3.1.1 Intended vulnerability

Time variability between different kinds of instructions can be used to map the canned set of source codes to a set of time observations.

As source code is interpreted, the challenge application sends SSL-encrypted packets containing updated register/memory values. A jump/branch instruction causes the interpreter to throw an exception resulting in an observable delay in its reply.

An attacker can observe another user's interaction, collect a set of time observables and use the pre-computed mapping to determine the source code run by the user.

4.7.2.4.2.3.1.2 R&D Performers

Through experimentation, the teams discovered that the initial values that seemed user-editable were discarded at the emulator's runtime, meaning each execution of an out#.s file followed the same instruction pattern and resulted in deterministic register value updates each run.

One team noted "Each program will take a certain amount of time, no matter how many times it is run. Thus, we just need to profile all ~500 programs with Profit and see how many collisions. If there are more than ~20 collisions, then its a NO. Otherwise, a YES. HOWEVER: After execute button is hit, server sends a push notification back with updates on the register values. Because we are constrained to interface with the program in the same manner as the example script which uses curl commands, EACH PUSH IS A DIFFERENT TCP CONNECTION. That means we can only detect up to 24ish pushes! We probably can't get enough timing information from this, since one program takes 1s per instruction"

As the clarifications given were that the user is to interact with the application in the same manner as the example script, in this case via curl where each was a separate connection, the intended vulnerability was rendered impossible with the number of operations given.

4.7.2.4.2.3.2 Wordshare Q005, AC Space

4.7.2.4.2.3.2.1 Intended Vulnerability

Hash collisions in the document name field can be used to trigger excessive disk usage. WordShare documents can be assigned aliases which can be made do collide with one another. Alias name collisions result in a call the WordShareDoc object's equals() causing check of whether or not the contents of the on-disk files related to the colliding names are equal. The equality results in an additional copy of a document on disk for each hash collision that occurs.

4.7.2.4.2.3.2.2 R&D Performers

The Researchers found an exploit that was in budget to produce many files that collectively could exceed the usage limit, but thought that "files", plural, was impossible given the following question 005: "Is there an algorithmic complexity vulnerability in space that would cause the challenge program to store a file with a logical size that exceeds the resource usage limit given the input budget?" As they explained their reasoning in the justification, the teams were given credit for their answer and this question was ruled Unintentionally Non-vulnerable.

4.7.2.4.2.3.3 Wordshare Question 047 (SC Time, Intended Vulnerable)

4.7.2.4.2.3.3.1 Intended Vulnerability

The application uses a backing trie structure to index stored secure and unsecure document contents. Wordshare allows users to modify the encoding scheme of a document and indexes both the secure and unsecure contents of the document in the same trie structure. When deleting a document, all unique trie nodes are removed until a node shared with another document is found. Time side channel in delete operations can be combined with the attacker's ability to control the encoding and the trie data structure to leak an encrypted word character-by-character. Attacker can create a matching unsecure version of a document to ensure that it shares as many nodes as possible with the secret one. Can then delete documents and use the time observations to determine characters of the secret encrypted word.

4.7.2.4.2.3.3.2 R&D Performers

The Researchers concluded that the encryption code was nonvulnerable. Some teams noted methods in the TrieNode code could have timing leaks and another thought times could depend on if a prefix is already stored in the triedata structure. They analyzed the encryption process and deemed it secure. One team mentioned there could be an attack adding elements to the Trie to overload the paths but did not find anything. None found a situation where the potential time differences were observable. No teams mentioned the delete function or its potential role in the vulnerability.

4.7.2.4.2.3.3.3 Control Team

The Control Team investigated if the search function had timing differences, but timings were consistent through their experimentation. Even if they found a leak, the Control Team thought it would be impossible to distinguish secret/non-secret words because they are in same trie. They also noted their attack from Wordshare Q002 is impossible without access to a secure document

5 Conclusion

The four-year STAC program researched approaches to identify and rule out Algorithmic Complexity and Side Channel vulnerabilities in Time and Space. The program began by measuring the capabilities of individual R&D performers but evolved to evaluate the combined performance of all the R&D teams. Ultimately, we concluded that based on the underlying assumption that the Control Team (Two Six Labs) represented the capabilities of the current state-of-the-art in the 5 STAC vulnerability categories, the STAC R&D performers advanced the state-of-the-art in the categories of Algorithmic Complexity vulnerabilities in Time and Space, and in Side Channel vulnerabilities in Time.

The STAC R&D performers were tasked with developing tools to perform semi-automated analysis on Java applications to aid analysts in identifying and ruling out vulnerabilities in these classes. The results of the final engagement were promising but there are remaining some remaining research tasks to help create transitionable capabilities from the results of the research developed by the R&D performers in the STAC program.

The tools created by the R&D performers are written to target Java applications, but the underlying research to identify and asses the strength of Side Channel and Algorithmic Complexity vulnerabilities can be adapted to analyze applications written in other languages.

The R&D performers' tools require analysts to provide a large amount of context and guidance to be successful. Analysts must manually take a large problem space and target these tools at relevant sections of code for the tools to be helpful. If the analyst provides too large a problem space, the tools may fail to work properly, if the analysis removes too much of the problem space, the tools will perform an accurate analysis of the reduced problem space, but that space will not be representative of the program as a whole.

The R&D performers' tools do not generally provide a binary response of vulnerable or non-vulnerable. Instead, these tools targeted at a manageable region of code, can identify potential conditions for vulnerability, flag patterns that indicate potentially vulnerable code, and with a sufficient part of the input space fixed, reason about the remaining space and identify and dynamically analyze inputs to this reduced space that result in excessive resource consumption or differential resource consumption that results in a reduction of the entropy of the secret – an indicator of a potential side channel vulnerability.

Finally, while the conclusions of the final engagement provide an assessment of the combined capabilities of the STAC tools and approaches, the tools are not integrated in any way. Instead the analyst is responsible for assessing which tools can perform which analysis, and how best to combine the outputs of multiple tools and pass those results as inputs to another tool or make conclusions about the vulnerability or non-vulnerability of the program under analysis.

Future work on these vulnerability categories could focus on an effective pruning of the problem space to enable some of the more targeted analysis approaches to be more effective without relying solely on the analyst to effectively prune the problem space. Efforts could be made to enable to create a more integrated capability that can perform analysis on entire applications and task different tools to perform analysis on different subproblems.

6 References

No External References.

Appendix A: Individual Engagement Responses

List of Tables

List of Equations

A.1 Engagement 1

The overall accuracy of the Blue Teams was 64.6% (63.1% when not including the control team) and the total combined AAD score for the Blue Teams was 1.09 (0.97 when not including the control team). If we weight the total accuracy according to AAD scores, we find the normalized total accuracy to be 65.3% (64.0% for the R&D Teams alone). Of the 82 questions attempted, 74 were answered “yes” and only 8 were answered “no”. This suggests that the teams focused on what they believed to be genuine vulnerabilities and went to verify a vulnerability as opposed to verify no vulnerability (which was expected as the latter is a much more difficult problem). Also, of the 82 questions attempted, 54 of them were AC in Time questions. This suggests that AC in Time vulnerabilities are more suitably handled by the Blue Teams’ tools.

The incorrect responses by the Blue Teams for both AC and SC questions fall into four categories. These categories are not identical in both AC and SC questions, but they mirror each other. For AC questions the incorrect Blue Team response categories are:

AC 17. Did not follow directions or violated the question definitions
AC 18. Failed to identify the complexity within the challenge program
AC 19. Failed to understand the input to the complexity control flow of the challenge program
AC 20. Failed to properly determine the complexity bounds of an identified complexity

For SC questions the incorrect Blue Team response categories are:

SC 17. Did not follow directions or violated the question definitions
SC 18. Failed to identify the secret and the secret location within the challenge program
SC 19. Failed to identify the side channel (processing conditioned on the secret value)
SC 20. Failed to properly analyze the side channel strength

The resulting distributions of incorrect responses for both AC and SC questions are shown in the two tables below.

Table A-1: Engagement 1 AC Responses

	Correct	Incorrect	AC 1	AC 2	AC 3	AC 4
AC Time	42	12	2	5	3	7
AC Space	6	6	0	1	1	4
Total	**48**	**18**	**2**	**6**	**4**	**11**

Table A-2: Engagement 1 SC Responses

	Correct	Incorrect	SC 1	SC 2	SC 3	SC 4
SC Time	4	5	1	0	2	3
SC Space	2	4	3	0	0	2
SC T & S	0	1	0	0	1	1
Total	**6**	**10**	**4**	**0**	**3**	**6**

From the tables above the Blue Teams had the most difficulty with analyzing identified complexities to determine the bounds of the complexities (11 incorrect) and properly identifying the complexities within the challenge program (6 incorrect). For side channel questions, the Blue Teams had the most difficult with analyzing and identifying the strength of discovered side channels (6 incorrect).
The following subsections describe in detail each performer's responses.

A.1.1 Northeastern University

A.1.1.1 NEU Overview

NEU had the same 5-member team throughout the 3 day (4 total sessions) of E1. They showed up late on the second day and were given a discounted start time (8:50 AM as opposed to 8:35 AM for the rest of the teams) due to the possibility of confusion as to when the engagement would start in the morning. This resulted in a total of 67.92 analyst hours (8.65 analyst days).

NEU had a total accuracy of 81.8% with answering a total of 11 questions (9 of which were answered correctly) after post engagement analysis. Of the 11 questions answered, 9 (7 correct) were AC in Time and 2 (both correct) were AC in Space. NEU did not attempt any side channel questions. NEU, like the other performers, focused on what they believed to be actual vulnerabilities. Of the 11 questions they answered, 10 were answered as "yes" and one was answered as "no". The following tables summarize their answers and their correctness based on the intended vulnerabilities (the raw answers from the engagement) and their answers after post-engagement analysis (the correct answers).

Table A-3: Engagement 1 NEU Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	0	0	0	0	0
SC in Time	0	0	0	0	0
SC in Space and Time	0	0	0	0	0
AC in Space	2	1	50	2	100
AC in Time	9	7	77.8	7	77.8
Total	**11**	**8**	**72.7**	**9**	**81.8**

Table A-4: Engagement 1 NEU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	7	7	**100**	6	**85.7**
Not Intended Vulnerable	4	1	**25**	3*	**75**
Yes Answer	10	7	**70**	8	**80**
No Answer	1	1	**100**	1	**100**

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

NEU's incorrect AC responses were due to either incorrectly following the directions or violating the question definitions (Question 019) or a failure to identify the complexities within the challenge program (Question 005)

As a note, NEU discovered the unintended vulnerability that exists in the TextCrunchr apps resulting from a highly compressed input. Also, NEU was one of the only teams to discover the intended zip bomb vulnerability in textcrunchr_1.

A.1.1.2 NEU Specific Results

A.1.1.2.1 Blogger

- **Questions 016 (AC Space, Intended Not Vulnerable, Answered No)**

  Northeastern University responded that their analysis tool identified 24 methods in the challenge program which could be used as a source of user input. They examined the loops that follow from the methods and tested the application under the given input budget. They concluded that there was not an AC Space vulnerability within the challenge program.

  Northeastern University correctly identified no AC Space vulnerability within the challenge program.

  Post-Engagement Analysis Ruling: Correct
- **Question 040 (AC Time, Intended Vulnerable, Answered Yes)**

  Northeastern University noticed that the java.io.PushbackInputStream: read method is used to construct the header values. They identified this method as a possible source for an AC Time. They manually analyzed the code and encountered a loop in the URIVerifier.verify method which is used to process a stack of URIElements. They recognized that by providing a URI with many sub-levels, they could exceed the resource usage limit.

  Northeastern University correctly identified the depth first search (DFS) vulnerability within the challenge program and provided an exploit script for the vulnerability which exceeded the resource usage limit.

  Post-Engagement Analysis Ruling: Correct

A.1.1.2.2 Image Processor

- **Question 005 (AC Time, Intended Vulnerable, Answered Yes)**

  The Northeastern University analyzer listed a set of loops in the challenge program which could be affected by user inputs. They manually inspected the code within the loops

and recognized that the cluster.controller.class extracted features of the training data. Northeastern University used images under the "example/images" folder and submitted the images to the application with different tags. They stated that their approach satisfied the input budget and exceeded the resource usage limit.

Northeastern University did not identify the location of the intended vulnerability in the Mathematics.intensify method. They claimed to have exceeded the resource usage limit by submitting images from the "example/images" folder to the application with different tags. The image processing challenge program folder stores training images in the "challenge_program/images" directory and example classification images in the "examples/classification_images" directory. Within the Docker container the training images in the "challenge_program/images" directory are moved to the "/var/lib/trainer/images" directory. The directory referenced by Northeastern University does not exist. If Northeastern University was referencing the training images, there is an unintended vulnerability which allows users to modify the "trainingSet.csv" file used to store the list of training images with their associated tags. Northeastern University did not describe this method which would enable a user to retag images and cause the challenge program to exceed the resource usage limit. If Northeastern University was referencing the example classification images within the "examples/classification_images" directory, there is an unintended vulnerability which allows users to add images to the training set of images. In order to prevent users from modifying the set of training images, the training images are stored within a directory which users cannot modify. The challenge program prepends a fixed file path to each image path read from the "trainingSet.csv" file. A user can add a path to the "trainingSet.csv" goes from the fixed file path to any image. This enables the user to modify the set of training images and to add multiple tags to the same image. Northeastern University did not describe this method which allows users to modify the set of training images and add multiple tags to the same image.

Northeastern University did not identify the intended AC Time vulnerability or any of the unintended AC Time vulnerabilities within the challenge program. Additionally they did not provided any evidence that they were able to exceed the resource usage limit. Northeastern University did not receive credit for this response.

Post-Engagement Analysis Ruling: Incorrect

A.1.1.2.3 Law Enforcement Database

- **Question 019 (AC Time Intended, Not Vulnerable, Answered Yes)**

  Northeastern University sent 110 search requests to the server. NEU started the timer prior to sending the first search request and stopped the timer after sending the 110th search request.

  The timing method used by NEU violates the resource usage limit instruction provided in the challenge question. The question states that the runtime is measured as the time interval between sending the last byte of the attack and the server sending the last byte of the response. It is the server processing time plus the network time for the server's responses to reach the client. The method proposed by Northeastern University violates this definition.

  Post-Engagement Analysis Ruling: Incorrect

A.1.1.2.4 TextCrunchr_1

- **Question 21 (AC Time, Intended Vulnerable, Answered Yes)**

Northeastern University identified that the user input to the challenge program is passed to the ZipDecompressor: decompress method. They identified that this loop will repeatedly unzip data from an archive but does not limit the depth of the archive. They concluded that provided a zip file that reproduces itself after zipping, you can cause the loop to run indefinitely.

Northeastern University correctly identified the zip bomb vulnerability within the challenge program.

Post-Engagement Analysis Ruling: Correct

A.1.1.2.5 TextCrunchr_2

- **Question 37 (AC Time, Intended Vulnerable, Answered Yes)**

  Northeastern University identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program. The intended vulnerability was a Hash Table which was vulnerable to hash collisions. While this vulnerability couldn't be triggered by a text file it can be triggered by submitting the file as a zip.

  Post-Engagement Analysis Ruling: Correct

A.1.1.2.6 TextCrunchr_3

- **Question 11 (AC Time, Intended Vulnerable, Answered Yes)**

  Northeastern University identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program. The intended vulnerability was a vulnerable sorting algorithm which has an O (n log (n)) runtime but can be caused to run in O (n^2) given an input file which is sorted prior to the quicksort call.

  Post-Engagement Analysis Ruling: Correct

A.1.1.2.7 TextCrunchr_5

- **Question 17 (AC Space, Intended Not Vulnerable, Answered Yes)**

  Northeastern University identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program.

  Post-Engagement Analysis Ruling: Correct

- **Question 24 (AC Time, Intended Not Vulnerable, Answered Yes)**

  Northeastern University identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program.

  Post-Engagement Analysis Ruling: Correct

A.1.1.2.8 TextCrunchr_6

- **Question 8 (AC Time, Intended Vulnerable, Answered Yes)**

Northeastern University identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

This was an unintended vulnerability in the challenge program. The intended vulnerability was in a sorting algorithm which when given inputs of a certain length incorrectly subdivides the sorting problem.

Post-Engagement Analysis Ruling: Correct

A.1.1.2.9 TextCrunchr_7

- **Question 33 (AC Time, Intended Vulnerable, Answered Yes)**

  Northeastern University identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file. Northeastern University stated that the exploit file they provided causes the Enigma component of TextCrunchr_7 to run slowly.

  This was an unintended vulnerability in the challenge program. The intended vulnerability was a vulnerable hash table. The table uses a red black tree but fails to balance the tree on puts and is vulnerable to DOS.

  Post-Engagement Analysis Ruling: Correct

A.1.2 University of Colorado Boulder

A.1.2.1 UC Boulder Overview

UC Boulder had a 6 person team for the first 2 days (3 sessions) and a 5 member team on the last day (1 session). This resulted in a total of 79.53 analyst hours (9.94 analyst days).

UC Boulder had a total accuracy of 30% with answering 10 questions (3 of which were answered correctly). UC Boulder only answered "yes" to questions and so these results generally represent UC Boulder's ability to identify vulnerabilities. Again, both the raw answers from the engagement and the corrected answers after post-engagement analysis are presented.

Table A-5: Engagement 1 UC Boulder Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	1	1	100	0	0
SC in Time	1	0	0	0	0
SC in Space and Time	0	0	0	0	0
AC in Space	3	0	0	0	0
AC in Time	5	5	100	3	60
Total	**10**	**6**	**60**	**3**	**30**

Table A-6: Engagement 1 UC Boulder Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	6	6	**100**	3	**50**
Not Intended Vulnerable	4	0	**0**	0	**0**
Yes Answer	10	6	**60**	3	**30**
No Answer	0	0	**0**	0	**0**

UC Boulder answered 8 AC questions in Engagement 1 with a 30 % accuracy. Of the 5 incorrect AC responses by UC Boulder, 3 (Blogger Question 016, GabFeed_1 Question 038, and Image Processor Question 005) were incorrect because UC Boulder failed to properly determine the complexity bounds of the identified complexities. For Image Processor Question 030 UC Boulder identified a complexity but failed to determine the bounds of the complexity, and failed to identify the vulnerable complexity within the challenge program. UC Boulder answered 2 SC questions both of which were incorrect. For GabFeed 3 Question 023, UC Boulder identified a method which did not contain a side channel and failed to identify the method which contained the side channel. For GabFeed 5 Question 001, UC Boulder identified a side channel which was not strong enough to satisfy the operational budget, and violated the passive observations component of the operational budget.

A.1.2.2 UC Boulder Specific Results

A.1.2.2.1 Blogger

- **Question 16 (AC Space, Intended Not Vulnerable, Answered Yes)**

  UC Boulder responded that their tool had identified a cubic complexity loop in the NanoHTTPD.decodeMultipartFormData method and that an append call in the loop could be exploited for an AC Space attack.

  The NanoHTTPD.decodeMultipartFormData method calls the java.util.regex.Pattern method at the quadratic level and the method only copies object references inside the cubic loop. There is no append method called within the loop. The algorithmic complexity identified by UC Boulder is bounded and does not exceed the resource usage limit. The method identified does not contain an intended AC Space vulnerability and UC Boulder did not provide an exploit script to validate the identified vulnerability.

  Post-Engagement Analysis Ruling: Incorrect
- **Question 40 (AC Space, Intended Vulnerable, Answered Yes)**

  UC Boulder responded that inside the function fi.iki.elonen.URIVerifier.verify() contains a high complexity loop (3) with a “push” inside. This vulnerability can be exploited by requesting URLs with a large nesting depth such as “localhost:8080/stac/stac/stac/stac…”.

  UC Boulder correctly identified the DFS vulnerability in the URIVerifier.verify method and provided a working exploit for the vulnerability.

  Post-Engagement Analysis Ruling: Correct

A.1.2.2.2 GabFeed_1

- **Question 38 (AC Space, Intended Not Vulnerable, Answered Yes)**

  UC Boulder stated that a user can send an unbounded number of new messages via 'NewMessageHandler.handlePost.' This will trigger an unbounded number of object constructions in 'GabThread.addMessage.' UC Boulder's tool determined that the user input affects the resource usage limit of the challenge program.

  UC Boulder correctly identified that the NewMessageHandler.handlePost method calls the GabThread.addMessage method. For each new message the GabThread.addMessage method will create several object constructors. They were incorrect in concluding that because there is no restriction within the application on the number of messages a user can post, that it is possible to exceed the resource usage limit by posting as many new messages as allowed by the input budget.

  The input budget is 400,000 bytes and the resource usage limit is 512 MB. A standard HTTP POST command is composed of a POST, a requested page, an HTTP version, a host, and a content-length field, the minimum size for a POST request to add a new message to a GabFeed threat is 60 bytes. If a user was able to craft a POST command using only the bare minimum required components and reduced the size to 40 bytes, with a budget of 400,000 bytes the user would be able to post 10,000 messages. We tested the GabFeed application by observing the memory usage of the application and posting 35,900 messages of the minimum size (1 byte). The GabFeed server's peak memory usage during this period was 286,544 KB of resident memory. This is less than the resource usage limit required for complexity.

  Post-Engagement Analysis Ruling: Incorrect

A.1.2.2.3 GabFeed_2

- **Question 6 (AC Time, Intended Vulnerable, Answered Yes)**

  UC Boulder responded that a user has control over the data structures that represent the database of the online portal. The vulnerable data structure is the hash map implemented in "com.cyberpointllc.stac.hashmap.HashMap." UC Boulder targeted their tool to the hash map that stores the chats in GabFeed (all reports from their tool could potentially apply to other cases as well where the user can both modify and query the data structure e.g. messages). Their tool reported paths from methods handleGet and handlePost inside the "com.cyberpointllc.stac.gabfeed.handler.ChatHandler" class to expensive operations of the "com.cyberpointllc.stac.hashmap.TreeNode" class (e.g. teeify and find). The latter methods perform operations that depend on the size of the internal tree. UC Boulder concluded that the challenge program contained an AC time vulnerability.

  UC Boulder correctly observed that the Hash Map structure used by GabFeed was vulnerable. The AC Time vulnerability in GabFeed 2 is located in the TreeNode class, but it is due to the fact that the tree structure is unbalanced. If a large message is posted which contains hash collisions the performance of the tree structure will become O (n^2). UC Boulder identified the location of the vulnerability within the challenge program. They did not explicitly report the complexity of the TreeNode class nor did they verify that the resource usage limit could be exceeded via the complexity. They provided the correct answer to the question and their justification is not incorrect.

  Post-Engagement Analysis Ruling: Correct

A.1.2.2.4 GabFeed_3

- **Question 23 (SC Time, Intended Vulnerable, Answered Yes)**

  UC Boulder stated that the SearchHandler.getTermsText method contains a loop which performs an expensive string concatenation operation for each special term.

  The challenge question asked if there was a Side Channel in time within the challenge program with which a third party can identify when a user has searched for a special search term. The SearchHandler.getTermsText method identified by UC Boulder is not called when a user searches for a term. The method is call when a user navigated to the search page. The method is used to identify and list the daily special search terms. The daily special search terms are listed on the search page for all users are not a secret.

  UC Boulder did not correctly identify the side channel.

  Post-Engagement Analysis Ruling: Incorrect

- **Question 36 (AC Time, Intended Vulnerable, Answered Yes)**

  UC Boulder responded that a user can create an unbounded number of messages in a chat. A user can trigger a call to the "ChatHandler.handleGet" method which may sort all the message of the user using the "Sorter.sort" method which is O (n*log (n)) where n is the size of the messages in the chat. UC Boulder's tool identified that this loop depends on user input. UC Boulder responded that the challenge program contained an AC time vulnerability.

  The AC Time vulnerability in GabFeed 3 is contained within the Sorter.sort method which is called in many places in the code. The sort method has an O (n log (n)) runtime in the benign case. The second mode of the sort method is triggered when the length of the list being sorted is divisible by 8. In the second mode of the method the runtime is worse than O (n^2).

  UC Boulder identified the method correctly but the O (n log (n)) mode they observed is the normal runtime mode. Within the input budget this mode will not exceed the resource usage limit. However; the path to the sort method identified by UC Boulder will allow the slow mode of the sort algorithm to be triggered. The sort method is called each time a new chat message is added, by sequentially adding 1 character chat messages, the slow mode will be triggered starting with the 12^{th} message that is added. Following the arithmetic sequence ($a_n = 12 + (n-1)*8$) when n is 44, the slow mode of the algorithm will be triggered and the resource usage limit will be exceeded. Because of the way in which they trigger the vulnerability UC Boulder receives credit for this response.

  Post-Engagement Analysis Ruling: Incorrect, need DARPA euling

A.1.2.2.5 GabFeed_5

- **Question 1 (SC Space, Intended Not Vulnerable, Answered Yes)**

  This question asked if there was a side channel in space within the challenge program from which a third party can discover the GabFeed username of a user involved in a private chat.

  UC Boulder responded that the UserHandler.handleGet method could be called with another user ID as the requested user. This method will call the ChatHandler.getChatContent for the other user. There is a branch in the method where one branch allocates a constant number of 'Link' objects while the other one allocates a number of 'Link' objects that is linear in the number of chats the requested user is involved in.

There are several issues with this response. The first is that the passive operations in the question are to observe the encrypted packets and timings thereof within a single user session. Observing the memory usage of the running server is not in the scope of the question. The second is that observing the memory of the running application to determine the number of chats a user is involved in is not feasible outside of the JVM. Observing memory usage by hooking into calls within the running JVM is outside of the scope of a side channel in space. Lastly UC Boulder states that the UserHandler.handleGet method calls the ChatHandler.getChatContent but the ChatHandler class does not have a getChatContents method. The method referenced by UC Boulder is the UserHandle.getChatContents this method creates the 'Link' objects referenced in the UC Boulder response.

Post-Engagement Analysis Ruling: Incorrect

A.1.2.2.6 Image Processor

- **Question 5 (AC Time, Intended Vulnerable, Answered Yes)**

  UC Boulder stated that an AC Time vulnerability could be triggered in the Image Processor program by issuing enough tag commands for the setTag method in the ClusterController class. When the 'cluster' command is called it will run 'cluster' for each tagged image and lastly for the input image. UC Boulder claimed that this would cause the resource usage limit to be exceeded.

  The challenge program, when run within the provided STAC Registry Docker container, uses the "/var/lib/trainer/images" directory to store training images. Within the container this directory cannot be modified by users. A user cannot change the training image set and can only apply tags to training images. If multiple tags are applied to an image, the 'cluster' command will proceed with the last tag for the image. The number of available tag operations is restricted to the number of images in the training image set which cannot be modified as per the description and within the provided container. If all the training images are tagged and the cluster command is called for a benign input, within the input budget, the resource usage limit will not be exceeded.

  There is an unintended vulnerability within the challenge program which could enable users to add duplicate tags to training images. After images are tagged, the tagged images are store in the trainingSet.csv file. This file can be modified by the user to add multiple tags to a give image. By adding a sufficient number of tags within the input budget a user can exceed the resource usage limit. There is another unintended vulnerability which could enable users to train/tag images not in the training set of images. Image paths listed in the trainingSet.csv file are prepended with a fixed path by the challenge program when the cluster command is called. Users can create a path to an image outside of the fixed training set from the fixed path appended by the challenge program. This would enable users to add images to the training set. The response provided by UC Boulder did not identify either of these possible methods for adding images to the training set.

  Post-Engagement Analysis Ruling: Incorrect, need DARPA ruling

- **Question 30 (AC Space, Intended Not Vulnerable, Answered Yes)**

  UC Boulder stated that because users control the training set of the application the user can increase the training set and cause the program to run out of memory.

  Referencing both the description and the provided container users cannot modify the training set. Referencing the unintended vulnerability description for Question 5 above, the

response provided by UC boulder did not provide a method which could enable users to increase the number of images in the training set.

There is an unintended vulnerability within the challenge program. The challenge programs accepts compressible image formats as inputs. After reading in input images, these images are converted to bitmaps for analysis. It is possible to create a highly compressible image and compress the image to a jpeg to be within the input budget. When the image is read in by the challenge program, the image will uncompressed in order to create a bitmap of the image. This could cause the challenge program to exceed the resource usage limit. This unintended vulnerability was not identified by UC Boulder.

Post-Engagement Analysis Ruling: Incorrect

A.1.2.2.7 TextCrunchr_6

- **Question 8 (AC Time, Intended Vulnerable, Answered Yes)**

  UC Boulder responded that the program implements an inefficient sorting method in the Sorter.ClasschangingSort class, the doIt1 method, which takes up 38.9% of the CPU time while processing an input file.

  This is the correct location of the vulnerability in the program. The sorting algorithm normal mode has an O (n log (n)) runtime but the slow mode has an O (n^2) runtime which can be used to exceed the resource usage limit within the input budget. UC Boulder did not specifically identify the two modes of the sorting algorithm or that input list lengths that are multiples of 3 cause the sort to run in the slow mode. UC Boulder did not provide an exploit script or a description of an exploit for this vulnerability.

  Post-Engagement Analysis Ruling: Correct

A.1.3 GrammaTech

A.1.3.1 GrammaTech Overview

GrammaTech had the same 4 person team for all 3 days. This resulted in 55.33 analyst hours (or 6.92 analyst days). GrammaTech answered a total of 10 questions with a 90% accuracy. Of the 10 that were attempted, 8 were AC in Time, 1 was AC in Space, and 1 was SC in Space. The only questions that GrammaTech correctly answered were AC questions. As with other performers, GrammaTech only answered “yes” to questions. The following tables summarize these results.

Table A-7: Engagement 1 GrammaTech Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	1	0	0	0	0
SC in Time	0	0	0	0	0
SC in Space and Time	0	0	0	0	0
AC in Space	1	0	0	1	100
AC in Time	8	7	87.5	8	100
Total	**10**	**7**	**70**	**9**	**90**

Table A-8: Engagement 1 GrammaTech Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	7	7	**100**	7	**100**
Not Intended Vulnerable	3	0	**0**	2*	**66.7**
Yes Answer	7	10	**70**	9	**90**
No Answer	0	0	**0**	0	**0**

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

As a note, of the 10 questions GrammaTech answered, they answered all 7 of the TextCrunchr app questions. Due to the unintended vulnerability found in TextCrunchr, GrammaTech's accuracy improved from 70% to 90% after post-engagement analysis. Also, GrammaTech said they found an unintended space based side channel in the Law Enforcement Database problem.

GrammaTech answered 9 AC questions all correctly and 1 SC question (L.E.D. Question 027). GrammaTech's single incorrect response was because GrammaTech did not properly analyze the strength of the SC they identified.

A.1.3.2 GrammaTech Specific Results

A.1.3.2.1 Blogger

- **Question 40 (AC Time, Intended Vulnerable, Answered Yes)**

  GrammaTech responded that Taint-sink identification pointed to a nested loop in the de-compilation of the Blogger challenge program. They identified a path from the ClientHandler.run to the URIVerifier.verify. When they inspected the line they noticed that this was an implementation of Non-Deterministic Finite Automata (NFA) for regular expression matching. They were able to use this knowledge to craft an input request to the server which exceed the resource usage limit.

  They correctly identified the vulnerability in the URIVerifier.verify method.

  Post-Engagement Analysis Ruling: Correct

A.1.3.2.2 Image Processor

- **Question 5 (AC Time, Intended Vulnerable, Answered Yes)**

  GrammaTech stated that Taint-sink identification points to 12 loop guards including one in the loop in com.stac.mathematics.Mathematics.exp which is a reimplementation of the exponent function. Taint-source identification pointed to 11 sources, including input underneath clusterAgainstTrainingSet. They ran the application with the provided blue.jpg image which showed a high cost and high execution counts for com.stac.mathematics.Mathematics.exp and Mathematics.factorial. Translation to C4B input suggests deep loop nesting but C4B didn't compute a useful bound but identified nesting as an issue. The Mathematics.exp loop is nested in a doubly-nested loop of the x and y coordinates. The loop performs filters.Intensify on the color values which results in very high iteration counts for some values of RGB-channel intensities. GrammaTech did

not confirm the problem with an input but they believed they could craft an input with same-color pixels that compress to a small jpg file to maximizes loop in Mathematics.exp

GrammaTech identified the location of the intended vulnerability in the challenge program. While they did not provide an exploit script their description of the exploit is enough to validate that it would work.

Post-Engagement Analysis Ruling: Correct

A.1.3.2.3 Law Enforcement Database

- **Question 27 (SC Space, Intended Not Vulnerable, Answered Yes)**

  GrammaTech responded that Taint-sink identification pointed to the loop in channelRead0 in UDPServerHandler. In the UDPServerHandler class the channelRead0 function contains a space side-channel leak. When answering a search request on the range between two public keys one allocation will be performed if and only if the range contains a restricted ID. Using binary search to restrict the search range while tracking memory usage of the server could allow them to pinpoint a restricted ID. GrammaTech proposed a method of performing a binary search over the range. Their proposed method would take log 2 (n) operations if n is the gap size between the smallest and largest public key. They state that IDs lie between 100,000 and 40,000,000. The range is actually 0 to 40,000,000; therefore their proposed search method would require 26 operations.

  Question 27 provides no passive operations for finding the side channel. Observing the memory usage of the challenge program is outside of the scope of the challenge question. If the memory was observable there is another issue with this response. The channelRead0 function in the UDPServerHandler class for search operations iterates through the returned IDs. Non-restricted IDs are returned to the user but in the case of a restricted ID the function throws a RestrictedAccessException. In handling the exception the function iterates through the IDs until a non-restricted ID is found. There are objects that are created within this loop: a string and a DSystemHandle object. These objects are small enough that the JVM does not have to allocate more memory. Since memory allocated within the JVM is observed externally as resident memory use and the JVM doesn't need to allocate more memory for these cases it would not be possible to observe this channel even if the question provided passive operations.

  Post-Engagement Analysis Ruling: Incorrect

A.1.3.2.4 TextCrunchr_1

- **Question 21 (AC Time, Intended Vulnerable, Answered Yes)**

  GrammaTech identified an unintended vulnerability in the challenge program. The guards on the input zip file do not prevent users from submitting a highly compressible input file.

  GrammaTech receives credit for this question but the intended vulnerability was a zip bomb vulnerability that would allow users to create an endless recursion loop within the zip file.

  Post-Engagement Analysis Ruling: Correct

A.1.3.2.5 TextCrunchr_2

- **Question 37 (AC Time, Intended Vulnerable, Answered Yes)**

GrammaTech identified an unintended vulnerability in the challenge program. The guards on the input zip file do not prevent users from submitting a highly compressible input file.

GrammaTech receives credit for this question but the intended vulnerability was a Hash Table which was vulnerable to hash collisions. While this vulnerability couldn't be triggered by a text file it can be triggered by submitting the file as a zip.

Post-Engagement Analysis Ruling: Correct

A.1.3.2.6 TextCrunchr_3

- **Question 11 (AC Time, Intended Vulnerable, Answered Yes)**

  GrammaTech identified an unintended vulnerability in the challenge program. The guards on the input zip file do not prevent users from submitting a highly compressible input file.

  GrammaTech receives credit for this question but the intended vulnerability was a vulnerable sorting algorithm which has an O (n log (n)) runtime but can be caused to run in O (n^2) given an input file which is sorted prior to the quicksort call.

  Post-Engagement Analysis Ruling: Correct

A.1.3.2.7 TextCrunchr_5

- **Question 17 (AC Space, Intended Not Vulnerable, Answered Yes)**

  GrammaTech identified an unintended vulnerability in the challenge program. The guards on the input zip file do not prevent users from submitting a highly compressible input file.

  Post-Engagement Analysis Ruling: Correct

- **Question 24 (AC Time, Intended Not Vulnerable, Answered Yes)**

  GrammaTech identified an unintended vulnerability in the challenge program. The guards on the input zip file do not prevent users from submitting a highly compressible input file.

  GrammaTech receives credit for these questions but the challenge program contained no intended vulnerabilities.

  Post-Engagement Analysis Ruling: Correct

A.1.3.2.8 TextCrunchr_6

- **Question 8 (AC Time, Intended Vulnerable, Answered Yes)**

  GrammaTech identified an unintended vulnerability in the challenge program. The guards on the input zip file do not prevent users from submitting a highly compressible input file.

  GrammaTech receives credit for this question but the intended vulnerability was in a sorting algorithm which when given inputs of a certain length incorrectly subdivides the sorting problem.

  Post-Engagement Analysis Ruling: Correct

A.1.3.2.9 TextCrunchr_7

- **Question 33 (AC Time, Intended Vulnerable, Answered Yes)**

GrammaTech identified an unintended vulnerability in the challenge program. The guards on the input zip file do not prevent users from submitting a highly compressible input file.

GrammaTech receives credit for this question but the intended vulnerability was a vulnerable hash table. The table uses a red black tree but fails to balance the tree on puts and is vulnerable to DOS.

Post-Engagement Analysis Ruling: Correct

A.1.4 Draper Labs

Draper Labs arrived late on the first day due to confusion on their part of the start time of the engagement. Due to this and issues setting up their server outside of the Draper network, they were given a discounted start time of 1:50PM (everyone else started at 1:03 PM). For all three days, Draper had the same 4 person team and this resulted in a total of 52.2 analyst hours (6.53 analyst days).

Draper answered a total of 5 questions, all of which were AC questions (3 Time and 2 Space). Of the questions they answered, Draper only correctly answered 1 (an AC in Space question) which results in a total accuracy of 20%.

Table A-9: Engagement 1 Draper Labs Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	0	0	0	0	0
SC in Time	0	0	0	0	0
SC in Space and Time	0	0	0	0	0
AC in Space	2	1	50	1	50
AC in Time	3	0	0	0	0
Total	**5**	**1**	**20**	**1**	**20**

Table A-10: Engagement 1 Draper Labs Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	3	0	**0**	0	**0**
Not Intended Vulnerable	2	1	**50**	1	**50**
Yes Answer	1	0	**0**	0	**0**
No Answer	4	1	**25**	1	**25**

As a note, Draper was one of the few teams to answer “no” on questions and, in fact, their answers had the highest proportion of “no” answers (80%) as compared to the other performers. Draper also submitted a total of half of the “no” answers.

Draper Labs answered 5 AC questions, 4 of them incorrectly. Three of Draper's incorrect responses were because they failed to properly determine the complexity bounds of the identified vulnerabilities.

A.1.4.1 Draper Labs Specific Answers

A.1.4.1.1 Blogger

- **Question 16 (AC Space, Intended Not Vulnerable, Answered No)**

  Draper Labs responded that the AProVE tool produces an integer transition system corresponding to program space allocation. The KoAT tool showed a single infinite loop which corresponds to the user-server interaction. Within the loop there is only one constant space allocation. The coefficients suggest that this allocation is far below what is required for a space vulnerability. This is the correct response given that there was no intended vulnerability.

  Post-Engagement Analysis Ruling: Correct

- **Question 40 (AC Time, Intended Vulnerable, Answered No)**

  Draper Labs stated that their tools identified one single loop for the program and each operation within the loop takes a constant amount of time with an upper bound of 10 calls to the sleep method. They concluded that this interaction happens with constant time and that no time vulnerability is present.

  Within the program the only included vulnerability is in the parsing of a provided URI by the URIVerifier.verify method. There was no included vulnerability within the user-server interaction beyond the URI verification. Draper Labs did not identify the included vulnerability.

  Post-Engagement Analysis Ruling: Incorrect

A.1.4.1.2 Graph Analyzer

- **Question 20 (AC Time, Intended Vulnerable, Answered No)**

  Draper Labs responded their Visualization tool identified the GreedyClustering class is the only GraphActor that can be invoked on the graph. They identified that per step the GreedyClustering has a runtime quadratic in the number of vertices in the graph. Draper Labs observed linear runtimes for both the colorChildren and calcGain methods. They observed linear runtime for the program with the size of the input and concluded that because the input size was constrained, it was not possible to exceed the resource usage limit.

  GreedyCluster's step method has O (n*m) complexity where n is the number of vertices and m is the number of identified clusters (this can change as GreedyCluster runs). It is bounded by O (n^2) in the worst case. ColorChildren and calcGain are both linear complexity with the number of vertices in the worst case. Draper Labs recognized that the user input which controlled the runtime was the number of vertices in a given graph. They did not recognize that this could be increased within the budget to cause the challenge program to exceed the resource usage limit. They incorrectly assumed that there was a linear correlation between the input file size and the number of vertices in the produced graph; this relationship is exponential.

  Post-Engagement Analysis Ruling: Incorrect

A.1.4.1.3 Image Processor

- **Question 5 (AC Time, Intended Vulnerable, Answered No)**

  Draper Labs stated that there were two halves to this program, the image processing segment of the code and the training set management/ clustering segment of the code. Their tools were able to show that both portions of the program terminate, but they were unable to find an upper bound on the runtime for complexity for the integer transition system representing the training set management/ clustering segment of the program. For feature extraction they were able to prove a quadratic upper bounds in the number of pixels in the image for color detection and they found a quartic bound for edge detection. They concluded that this seems in line with the runtime complexity of image recognition algorithms.

  Draper Labs did not identify that Mathematics.intensify function which controls the pixel value set by the BufferedImage.getRGB is controlled by the user input and that a user can cause the output of intensify function to be large. Additionally, Draper labs pointed to the location of an unintended vulnerability but did not recognize that it could be used to exceed the resource usage limit. They identified a quadratic upper bounds in the number of pixels in the image for color detection. The quadratic method identified only has one mode, but given a highly compressed input image, the challenge program can be made to exceed the resource usage limit within the single mode of the color detections component of the challenge program.

  Post-Engagement Analysis Ruling: Incorrect

- **Question 30 (AC Space, Intended Vulnerable, Answered Yes)**

  Draper Labs stated that they observed multiple image files being created during processing and that for a sufficiently large number of images the resource usage limit could be exceeded

  The number of images is bound by the number of training images + 1 (image being processed). Given that the number of training images is fixed the provided resource usage limit cannot be exceeded by this method. Additionally Draper Labs did not provide a method by which the number of training images could be increased.

  Post-Engagement Analysis Ruling: Incorrect

A.1.5 University of Utah

A.1.5.1 Utah Overview

Utah had the same 6 person team all three days. On the third day, Utah showed up late and was given a discounted start time of 8:40 AM (as opposed to 8:32 AM for everyone else). This resulted in 82.2 total analyst hours (10.28 analyst days).

Utah answered a total of 2 questions, both of which were AC in Time questions, and correctly answered only 1 of them. This resulted in an overall accuracy of 50%. Also, Utah only answered "yes" to questions.

Table A-11: Engagement 1 Utah Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	0	0	0	0	0
SC in Time	0	0	0	0	0
SC in Space and Time	0	0	0	0	0
AC in Space	0	0	0	0	0
AC in Time	2	1	50	1	50
Total	**2**	**1**	**50**	**1**	**50**

Table A-12: Engagement 1 Utah Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	1	1	**100**	1	**100**
Not Intended Vulnerable	1	0	**0**	0	**0**
Yes Answer	2	1	**50**	1	**50**
No Answer	0	0	**0**	0	**0**

A.1.5.2 Utah Specific Answers

A.1.5.2.1 Law Enforcement Database

- **Question 19 (AC Time, Intended Not Vulnerable, Answered Yes)**

  University of Utah responded that given a restricted ID, a GETRECORD request for the restricted ID will cause the client to hang indefinitely.

  This is not the case, querying the client with a GETRECORD request for a restricted ID will result in a response of "?" in less than 30 seconds (the resource usage limit). This is not an algorithmic complexity within the challenge program.

  Post-Engagement Analysis Ruling: Incorrect

A.1.5.2.2 TextCrunchr_1

- **Question 21 (AC Time, Intended Vulnerable, Answered Yes)**

  University of Utah crafted an input zip file which is smaller than the input budget but expands to 4 13 MB text files. Processing these files causes the challenge program to exceed the resource usage limit.

  Post-Engagement Analysis Ruling: Correct

A.1.6 Iowa State University

A.1.6.1 Iowa State Overview

Iowa State had a 4 person team on the first day (1 session) and a 5 person team for the second and third days (3 session). This resulted in a total of 65.72 analyst hours (8.21 analyst days).

Iowa State answered "yes" to two questions, both AC in time questions, with an accuracy of 100%. It was found during post engagement analysis that both of these were in fact answered correctly and so their 100% accuracy is maintained

Table A-13: Engagement 1 ISU Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	0	0	0	0	0
SC in Time	0	0	0	0	0
SC in Space and Time	0	0	0	0	0
AC in Space	0	0	0	0	0
AC in Time	2	2	100	1	50
Total	**2**	**2**	**100**	**1**	**50**

Table A-14: Engagement 1 ISU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	2	2	**100**	1	**50**
Not Intended Vulnerable	0	0	**0**	0	**0**
Yes Answer	2	2	**100**	1	**50**
No Answer	0	0	**0**	0	**0**

A.1.6.2 Iowa State Specific Answers

A.1.6.2.1 Image Processor

- **Question 5 (AC Time, Intended Vulnerable, Answered Yes)**

  Iowa State responded that there was an AC Time vulnerability within the number of training images. They identified that the initialization process will call the command 'train' for all images in the specified training folder containing all the training images. The 'train' image will tag the input image with the user-specified tag. The tagging process reads the already trained/tagged images from the excel sheet trainingSet.csv. The process then fills a map with the images and their tags as key, value pairs. The new input image and its specified tag are added to the map. The key, value pairs are written back to the excel sheet. Iowa State recognized that the vulnerability occurs when the number of

training images are large. For a large number of tagged images within the input budget, the resource usage limit will be exceeded.

The description of the challenge program states that users cannot add new images to the training set. The delivery of the challenge program in the Docker container does not permit users to add new images to the training set. Iowa State did not identify the intended vulnerability in the challenge program. Iowa State did identify that the trainingSet.csv file is used to read in images when the cluster command is called. The challenge program appends a fixed path to the image paths referenced in the trainingSet.csv file. A user can edit this file and provide a path such that the appended fixed path will direct the challenge program to any file provided by the user. Iowa State did not identify this method which can enable users to add images to the training set.

Post-Engagement Analysis Ruling: Incorrect

A.1.6.2.2 TextCrunchr_1

- **Question 21 (AC Time, Intended Vulnerable, Answered Yes)**

  Iowa State discovered that the challenge program input guards to limit the input file size could be circumvented by providing a highly compressible input file as a compressed input file.

  Iowa State provided an input file which was within the input budget and caused the challenge program to exceed the resource usage limit.

  Post-Engagement Analysis Ruling: Correct

A.1.7 University of Maryland

A.1.7.1 UMD Overview

UMD answered the most questions (and had the most correct answers) of the R&D Teams. They had a 3 person team on the first day and 5 member teams on the second and third days. No team was identical between two different days (though some members of the team were present all days). This resulted in 62.27 total analyst hours (7.78 analyst days).

UMD was able to answer 15 questions with 11 of those being answered correctly resulting in an overall accuracy of 73.3%. Of the 15 UMD answered, 8 were AC in Time (7 AC Time correctly answered) which suggests, like others, AC in Time questions were the easiest for the teams to currently answer. UMD attempted answered both “yes” and “no” to questions, but had a 0% accuracy on the 2 questions they answered “no”. Their results are detailed in the tables below.

Table A-15: Engagement 1 UMD Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	1	1	100	1	100
SC in Time	4	2	50	1	25
SC in Space and Time	0	0	0	0	0
AC in Space	2	0	0	1	50
AC in Time	8	7	87.5	8	100
Total	**15**	**10**	**66.7**	**11**	**73.3**

Table A-16: Engagement 1 UMD Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	12	10	**83.3**	9	**75**
Not Intended Vulnerable	3	0	**0**	2*	**66.7**
Yes Answer	13	10	**76.9**	11	**84.6**
No Answer	2	0	**0**	0	**0**

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

A.1.7.2 UMD Specific Results

A.1.7.2.1 Blogger

- **Question 40 (AC Time, Intended Vulnerable, Answered Yes)**

  University of Maryland identified that the URIVerifier.verify method can be provided an invalid URI that causes it to loop excessively. They identified with taint analysis that the argument is attacker-controlled.

  Post-Engagement Analysis Ruling: Correct

A.1.7.2.2 GabFeed_1

- **Question 10 (SC Time, Intended Vulnerable, Answered No)**

  UMD identified that the ModPow.modPow method had a consistent runtime which made it unlikely that a side channel in time was present within the challenge program that could leak the server's private key.

  UMD did not identify that within the modPow method the algorithm iterates through the server's private key character by character. UMD's response is incorrect.

  Post-Engagement Analysis Ruling: Incorrect

A.1.7.2.3 GabFeed_2

- **Questions 32 (SC Time, Intended Not Vulnerable, Answered Yes)**

UMD identified by taint analysis that the base argument in the ModPow.modPow was user controlled. They stated that the method leaks the server's key because the runtime differs based on the exponent argument which is the server's private key.

UMD identified a side channel which was not intended and not present. They did not provide an exploit for this side channel. The response by UMD is incorrect.

Post-Engagement Analysis Ruling: Incorrect

A.1.7.2.4 Graph Analyzer

- **Question 34 (AC Space, Intended Vulnerable, Answered No)**

  UMD responded that the ".dot" input files are restricted by the graph/dot/DotFilter and that output objects are one-to-one with the input; this bounds the size of the ps output. The PNG output is fixed to 2000x2000 pixels which is insufficient to exceed the resource usage limit.

  The PNG output is bound at 64 bits per pixel; 32 MB. This will not exceed the resource usage limit. The ps output is bound by the number of vertices defined in the input file. UMD incorrectly assumes that there is a linear correlation between the bytes of input and the number of graph components (vertices and edges) defined by the input. This is not the case, this relationship is exponential. UMD's response is incorrect.

  Post-Engagement Analysis Ruling: Incorrect

A.1.7.2.5 Image Processor

- **Questions 5 (AC Time, Intended Vulnerable, Answered Yes)**

  UMD constructed a PNG file which is within the input budget but contains 900 million pixels. This image causes the challenge program to exceed the resource usage limit.

  UMD did not identify the intended vulnerability within the Mathematics.intensify method, but UMD recognized that the Intensify.filter method iterates through the submitted image pixel by pixel within a nested for loop. UMD recognized that there were no guards on the number of pixels within the image only on the size of the image and crafted an input PNG file of 30,000 pixels by 30,000 pixels. This file is within the input budget and cause the challenge program to exceed the resource usage limit.

  Post-Engagement Analysis Ruling: Correct

A.1.7.2.6 Law Enforcement Database

- **Question 26 (SC Time, Intended Vulnerable, Answered Yes)**

  UMD identified that the channelRead0 function is controlled by the user when a user submits a request. They identified that the method loops over each id within the range and checks whether the id is restricted. They stated that there is a potential for a timing attack because the runtime for the same number of unrestricted IDs changes when there is a restricted ID in the range.

  This is the correct vulnerability within the challenge program. Given that UMD did not provide an efficient exploit, using the vulnerability identified it is possible to identify the secret in 49 operations which is less than the operational budget. UMD receives credit for this response.

  Post-Engagement Analysis Ruling: Correct

A.1.7.2.7 SnapBuddy_2

- **Question 15 (SC Time, Intended Vulnerable, Answered Yes)**

  UMD identified that the ModPow.modPow method leaks the server's key because it's running time differs based on the exponent argument (the server's private key).

  The side channel identified by UMD could be used to determine the length of the server's private key. The server's private key is a 64 bit number, this method could be used to resolve the value of the server's private key within 2^{64} operations after the length of the key is determined. This is exceeds the operational budget for the question. The side channel which can be used to identify the server's private key within the operations budget is that there is a runtime difference between the cases where the server's nth bit is 0 verses when the n^{th} bit is one. This was not identified by UMD.

  Post-Engagement Analysis Ruling: Correct

A.1.7.2.8 SubSpace

- **Question 42 (SC Space, Intended Vulnerable, Answered Yes)**

  UMD responded that one could use two attacking users moving around while hailing each other to detect when a hail does not arrive. This method can be used to triangulate the attacked user's location.

  The proposed method by UMD requires 93 operations to identify the user's location within the range specified by the question of 1e-4 degrees. An attacker can place a user at the center of the grid and moving a second user to each of the corners of the grid. At each corner the user at the corner can send a hail. If the hail reaches the user at the center of the grid then the corresponding quadrant is empty. This method can be used to divide the range by 4 on each step. It requires 22 steps to identify the location of the user within 1e-4 latitude and longitude. The first step requires 5 operations, each subsequent step requires 4 operations; this is a total of 89 operations. It requires 4 operations to register and set location of the users added by the attacker. The 93 total operations required for this to work is within the operational budget.

  Post-Engagement Analysis Ruling: Correct, but needs DARPA approval

A.1.7.2.9 TextCrunchr_1

- **Question 21 (AC Time, Intended Vulnerable, Answered Yes)**

  UMD identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program. The intended vulnerability was a zip bomb vulnerability that would allow users to create an endless recursion loop within the zip file.

  Post-Engagement Analysis Ruling: Correct

A.1.7.2.10 TextCrunchr_2

- **Question 37 (AC Time, Intended Vulnerable, Answered Yes)**

  UMD identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

This was an unintended vulnerability in the challenge program. The intended vulnerability was a Hash Table which was vulnerable to hash collisions. While this vulnerability couldn't be triggered by a text file it can be triggered by submitting the file as a zip.

Post-Engagement Analysis Ruling: Correct

A.1.7.2.11 TextCrunchr_3

- **Question 11 (AC Time, Intended Vulnerable, Answered Yes)**

  UMD identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program. The intended vulnerability was a vulnerable sorting algorithm which has an O (n log (n)) runtime but can be caused to run in O (n^2) given an input file which is sorted prior to the quicksort call.

  Post-Engagement Analysis Ruling: Correct

A.1.7.2.12 TextCrunchr_5

- **Question 17 (AC Space, Intended Not Vulnerable, Answered Yes)**

  UMD identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program.

  Post-Engagement Analysis Ruling: Correct

- **Question 24 (AC Time, Intended Not Vulnerable, Answered Yes)**

  UMD identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program.

  Post-Engagement Analysis Ruling: Correct

A.1.7.2.13 TextCrunchr_6

- **Question 8 (AC Time, Intended Vulnerable, Answered Yes)**

  UMD identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

  This was an unintended vulnerability in the challenge program. The intended vulnerability was in a sorting algorithm which when given inputs of a certain length incorrectly subdivides the sorting problem.

  Post-Engagement Analysis Ruling: Correct

A.1.7.2.14 TextCrunchr_7

- **Question 33 (AC Time, Intended Vulnerable, Answered Yes)**

  UMD identified that a highly compressible input file could bypass the input guards of the challenge program and cause it to exceed the resource usage limit. They provided an exploit file.

This was an unintended vulnerability in the challenge program. The intended vulnerability was a vulnerable hash table. The table uses a red black tree but fails to balance the tree on puts and is vulnerable to DOS.

Post-Engagement Analysis Ruling: Correct

A.1.8 Vanderbilt University

A.1.8.1 Vanderbilt Overview

Vanderbilt had the same 5 person team all 3 days resulting in 69.17 total analyst hours (8.65 analyst days).

Vanderbilt answered a total of 10 questions with 6 of those being correctly answered. This results in an overall 60% accuracy. Of the 10 answered, 7 were AC in Time which again suggests that the performers' tools are more adequately geared towards AC in Time questions. Vanderbilt answered both "yes" and "no" to questions (9 yes and 1 no), but did not get the "no" question correct. The following tables present their results.

Table A-17: Engagement 1 Vanderbilt Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	1	0	0	0	0
SC in Time	1	1	100	0	0
SC in Space and Time	0	0	0	0	0
AC in Space	1	0	0	0	0
AC in Time	7	5	71.4	6	85.7
Total	**10**	**6**	**60**	**6**	**60**

Table A-18: Engagement 1 Vanderbilt Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	7	6	**85.7**	5	**71.4**
Not Intended Vulnerable	3	0	**0**	1*	**33.3**
Yes Answer	9	6	**66.7**	6	**66.7**
No Answer	1	0	**0**	0	**0**

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

As a note, Vanderbilt found the unintended vulnerability in the textcrunchr app. However, they also answered the wrong question for the Law Enforcement Database SC questions (they seem to have answered "given an ID, can you discover whether or not it is restricted" as opposed to the question asked of "can you find the restricted ID's within the allotted budget"). Thus, while their overall score did not change between the raw answers and the post-engagement analysis, the scores broken down between the different types of questions and answers did changed.

A.1.8.2 Vanderbilt Specific Results

A.1.8.2.1 Graph Analyzer

- **Question 34 (AC Space, Intended Vulnerable, Answered No)**

  Vanderbilt determined that the near-worst-case input is a complete graph with randomly positioned nodes. This results in the largest possible PNG file. They generated sparse input graphs and observed a linear relationship between the input and output file sizes. They concluded that the challenge program did not contain an AC space vulnerability.

  The PNG and PS output file formats are linear in complexity with the number of subgraphs. The PNG format overwrites repeated subgraphs and is bound by 16 MB per image. The PNG format cannot be used to exceed the space resource usage limit. Vanderbilt made the correct conclusion about the PNG format but was incorrect is their assumption about the correlation between the node positions and the resulting file size.

  Post-Engagement Analysis Ruling: Incorrect

A.1.8.2.2 Law Enforcement Database

- **Question 19 (AC Time, Intended Not Vulnerable, Answered Yes)**

  Vanderbilt stated by adding 5000 elements they were able to exceed the resource usage limit on a search operation over the interval. By analyzing the program using the callgraph and control-flow graph of the server with the loop count metric, they identified areas where a time complexity vulnerability could be located.

  The provided budget of 100 bytes allows for 198 insert operations and 1 search operations. The method proposed by Vanderbilt exceeds this budget.

  Post-Engagement Analysis Ruling: Incorrect
- **Question 26 (SC Time, Intended Vulnerable, Answered Yes)**

  Vanderbilt responded that there is an SC Time vulnerability within the challenge program which allows the attacker to verify if a candidate ID is restricted with one operation. By inserting a candidate ID and timing the response the attacker can determine if the ID is restricted or not.

  The secret is not whether a given candidate ID is restricted. The secret is identifying a specific restricted ID within the provided database. This is not a valid response to this question.

  Post-Engagement Analysis Ruling: Incorrect
- **Question 27 (SC Space, Intended Not Vulnerable, Answered Yes)**

  Vanderbilt responded that by inserting an ID and observing whether or not the ID is present in the subsequent search over the interval containing the ID, an attacker can discover whether or not a candidate ID is restricted.

  The secret is not whether a given candidate ID is restricted. The secret is identifying a specific restricted ID within the provided database. This is not a valid response to this question. To apply this logic over the interval of possible operations would be to do a brute force search over the range of possible keys. This would require more operations than the operational budget allows for.

  Post-Engagement Analysis Ruling: Incorrect

A.1.8.2.3 TextCrunchr_1

- **Question 21 (AC Time, Intended Vulnerable, Answered Yes)**
  Vanderbilt discovered an unintended vulnerability within the challenge program. They identified that the challenge program can be provided a highly compressible file which causes the challenge program to exceed the resource usage limit within the input budget.
  Post-Engagement Analysis Ruling: Correct

A.1.8.2.4 TextCrunchr_2

- **Question 37 (AC Time, Intended Vulnerable, Answered Yes)**
  Vanderbilt discovered an unintended vulnerability within the challenge program. They identified that the challenge program can be provided a highly compressible file which causes the challenge program to exceed the resource usage limit within the input budget.
  Post-Engagement Analysis Ruling: Correct

A.1.8.2.5 TextCrunchr_3

- **Question 11 (AC Time, Intended Vulnerable, Answered Yes)**
  Vanderbilt discovered an unintended vulnerability within the challenge program. They identified that the challenge program can be provided a highly compressible file which causes the challenge program to exceed the resource usage limit within the input budget.
  Post-Engagement Analysis Ruling: Correct

A.1.8.2.6 TextCrunchr_5

- **Question 24 (AC Time, Intended Not Vulnerable, Answered Yes)**
  Vanderbilt answered yes to this question and appears to have identified the unintended vulnerability within the TextCrunchr apps regarding highly compressed inputs. While this is not the intended vulnerability, it is recommended that credit be given for this discovery.
  Post-Engagement Analysis Ruling: Correct

A.1.8.2.7 TextCrunchr_6

- **Question 8 (AC Time, Intended Vulnerable, Answered Yes)**
  Vanderbilt discovered an unintended vulnerability within the challenge program. They identified that the challenge program can be provided a highly compressible file which causes the challenge program to exceed the resource usage limit within the input budget.
  Post-Engagement Analysis Ruling: Correct

A.1.8.2.8 TextCrunchr_7

- **Question 33 (AC Time, Intended Vulnerable, Answered Yes)**
  Vanderbilt discovered an unintended vulnerability within the challenge program. They identified that the challenge program can be provided a highly compressible file which causes the challenge program to exceed the resource usage limit within the input budget.
  Post-Engagement Analysis Ruling: Correct

A.1.9 Invincea (Control Team)

A.1.9.1 Invincea Overview

Invincea was the control team and had the same 5 person team all 3 days. This resulted in 69.17 analyst hours (8.65 analyst days). They answered the greatest number of questions (17 in all) with 11 of them being correct (64.7% accuracy). Invincea answered "yes" to every question and a majority of the questions answered were AC in Time questions. The following tables summarize their results.

Table A-19: Engagement 1 Invincea Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Corrected Number Correct	Corrected Accuracy (%)
SC in Space	2	1	50	1	50
SC in Time	3	2	66.67	2	66.7
SC in Space and Time	1	1	100	0	0
AC in Space	1	0	0	1	100
AC in Time	10	8	80	8	80
Total	**17**	**12**	**70.6**	**12**	**70.6**

Table A-20: Engagement 1 Invincea Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Corrected Number Correct	**Corrected Accuracy (%)**
Vulnerable	12	12	**100**	10	**83.3**
Not Intended Vulnerable	5	0	**0**	2*	**40**
Yes Answer	17	12	**70.6**	12	**70.6**
No Answer	0	0	**0**	0	**0**

*Includes correct "yes" answers for unintended vulnerabilities on questions that were not intended to be vulnerable.

As a note, Invincea found the unintended vulnerability in the TextCrunchr app and have provided exploit scripts for many of their answers. However, some of their answers were out of scope or incomplete. As an example, they are the only ones to have tackled to SC in Time and Space question (the search side channel in GabFeed_3), however their answer only includes justification for the space part and is not be a complete answer. These were the main reasons behind the changes from the raw answers to the corrected answers from post-engagement analysis.

A.1.9.2 Invincea Specific Answers

A.1.9.2.1 Blogger

- **Question 40 (AC Time, Intended Vulnerable, Answered Yes)**

  Invincea identified that in the URIVerifier.verify method, a long URI of the same character will trigger cause an AC Time vulnerability to be triggered. They also provided an exploit for this.

  Invincea identified the only included vulnerability correctly.

Post-Engagement Analysis Ruling: Correct

A.1.9.2.2 GabFeed_1

- **Question 7 (AC Time, Intended Vulnerable, Answered Yes)**

  Invincea identified that the LineBreak class in the message renderer is vulnerable to an algorithmic complexity attack. They identified that a large collection of very small tokens forces the string splitter to iterate of the entirety of the input numerous times in nested loops. They provided an exploit script for this vulnerability.

  Invincea correctly identified the vulnerability.

  Post-Engagement Analysis Ruling: Correct

- **Question 10 (SC Time, Intended Vulnerable, Answered Yes)**

  Invincea identified that there is a side channel in the server authentication of the application. The user can pass a challenge for the server to authenticate itself with. The authentication is a number of big number operation. The verification process starts with a big number of 1 then loops through the bits of the server's private key. For each bit it squares the initial value, then if the bit it 1, multiplies it by the client key. Invincea recognized that iterating backwards through the private server key, the operation performed when the bit is 0 is s^2 and when the bit is one it is the $client_key*$s^2$. Invincea provided the method by which this vulnerability could be used to leak the server's secret key. By selecting a client key to submit such that N operations arrive at 1 or 0 for each bit a malicious user can sample enough times to leak the server's secret key.

  Invincea correctly identified the location of key side channel vulnerability in the ModPow class. The ModPow class is called when the WebServer calls the LoginHandler which calls the Key ExchangeServer which then calls ModPow.modPow. The ModPow.modPow method takes arguments of the user's submitted public key, the server's private key, and a big integer number M. The algorithm then creates a BigInteger s = 1; the method iterates through the bits of the server's private key. The s variable is set to s^2 % M. If the current bit is 1, s = OptimizeMultiplier_fast with arguments of s*user public key % M. If the bit is 0 no change is made to the value of s. The timing difference between these two cases leaks the server's secret key and allows a malicious user to impersonate the server.

  Post-Engagement Analysis Ruling: Correct

A.1.9.2.3 GabFeed_2

- **Question 6 (AC Time, Intended Vulnerable, Answered Yes)**

  Invincea responded that the challenge program contained a vulnerability in the Hash Map hashing function which in cases of collisions behaves as a linked list. Invincea provided a malicious script which exploited this vulnerability.

  Invincea correctly identified the vulnerability. The challenge program contained a vulnerable hash table structure which on a large post will run in O (n^2).

  Post-Engagement Analysis Ruling: Correct

A.1.9.2.4 GabFeed_3

- **Question 36 (AC Time, Intended Vulnerable, Answered Yes)**

Invincea responded that the LineBreak.searchHelper function contained an AC vulnerability. They reported that they generated inputs which took 60 seconds to respond.

Invincea did not show that they were able to exceed the resource usage limit using this unintended vulnerability. The intended vulnerability in challenge program is contained within the Sorter.sort method which was not identified by Invincea.

Post-Engagement Analysis Ruling: Incorrect

- **Question 39 (SC Time/Space, Intended Vulnerable, Answered Yes)**

  Invincea stated that there is a side channel in gabfeed_3.jar for searching for special terms. When searching for a special term, the user receives a blurb describing the term. An attacker can permute all special terms to receive their blurbs. The blurbs are all different lengths [1] and are prepended to all search results. An attacker has access to the terms. Searching for these terms in succession will let her collect all description lengths, and the length of search results in case there are message or thread results that match them also. For easier analysis, the attacker can record her own traffic to correlate term description size to encrypted packet size. After this is collected, the attack becomes simple: watch for encrypted traffic and match the size of the results to the recorded lengths. Some terms are close enough in size (such as dyslexia and lupus) where the attack might not succeed, but there are enough large deltas to achieve 90% success rate.

  Invincea properly identified the correlation between the packet size and the corresponding special search term. Given that a user only searched for special terms this method works to identify the term. But if a user does any other operation, it is possible for that operation's packet size to collide with the sizes of the special terms. This would obscure the size only side channel identified. The time component of the side channel is necessary to remove the collisions and identify the whether a user search was for a special term. Given a budget of 6 operations and the current phrasing of the question it would be possible to answer the question using just a space based side channel. However because the budget is 5 in the worst cast case where there is a collision between the user operation and a special search term packet size on the first second and third user operation, resolving it assuming only one user operation in the third operation by the user can collide with a special search term (best case), the probability of success would be 50% which is less than the required 90%. Additionally the question asks for a side channel in space and time.

  Proposed ruling: Incorrect, need DARPA ruling

A.1.9.2.5 Law Enforcement Database

- **Question 19 (AC Time, Intended Not Vulnerable, Answered Yes)**

  Invincea provided an exploit script which sends multiple packets which in total are within the input budget limit. They sum the total time of all of the requests as one value in order to exceed the resource usage limit.

  The timing definition provided in the question is the time interval between sending the last byte of the attack and the server sending the last byte of the response. Timing 180 insert operations and one search operation does not count. The time of the single search operation did not exceed the budget. Invincea does not receive credit for this response.

  Post-Engagement Analysis Ruling: Incorrect

- **Question 26 (SC Time, Intended Vulnerable, Answered Yes)**

  Invincea stated that there is a side channel in time in the challenge program from which a third party can determine the value of any single restricted ID currently in the database.

An attacker can leak the value of any single restricted DI (sic) in the database in under 500 operations by exploiting the search command and mearing the time for the server response. The structure of the ID's is flattened to a list when the search command is called. The list is iterated through and if an ID number is not restricted, some operations are called and it is added to the final return list. If the ID is a restricted ID, several other operations are called (logs are made, an exception is triggered, etc.) which causes the timing of the search to change.

Invincea did not provide an exploit script but explained an exploit. They stated that an attacker can use this timing difference to serve of several list in ranges: [x, y], [x+1, y+1] …. The timing of each search is recorded. The attacker can for each search increment by one and observe the timing difference between the new and old interval. The attacker can increment the range by one until a search is reached which takes longer than the prior search times.

There are several issues with this. The server doesn't process a request and send the response at the end. The sever replies to the search request one packet at a time. The server fetches all the ID within a requested search range. The server then iterates through this list. If an ID is not restricted then the ID is sent to the client. If an ID is restricted then an error is thrown and the server iterates through the ID list until it finds a non-restricted ID. Getting the full response time for a search request would require a user to subtract the last packet received time from the first. The differential error of timing between each subsequent packet of the response would propagate. By the last packet the error of the time unit measured from the first to the last packet received for a request would make it impossible to resolve this side channel if it were present.

This method is equivalent to a brute force search over the interval which if it were possible with this method would require n operations where n is the minimum possible index – maximum possible index – known public keys. This number is far greater than the provided budget.

Post-Engagement Analysis Ruling: Correct, need DARPA ruling

- **Question 27 (SC Space, Intended Not Vulnerable, Answered Yes)**

  Invincea responded that there is a side channel in space in the challenge program. This side channel exists in the same manner as question 26 response. To make it a space side channel Invincea recognized that in the case of a non-restricted ID a log file is sent to port 6666 but for restricted IDs the log is written to a file. Invincea proposes to observe the size of this log file for the side channel.

  The challenge question provides no passive operation and observing the size of 'log' files is a passive operation. As per the description the logs contained in the files directory are out of scope.

  Post-Engagement Analysis Ruling: Incorrect

A.1.9.2.6 SnapBuddy_3

- **Question 22 (AC Time, Intended Vulnerable, Answered Yes)**

  Invincea responded that there was an AC Time vulnerability in the challenge program. Each image filter is a time consuming process and you can add arbitrarily many filters to run on a given image. They identified the vulnerability in the SnapServiceImpl.modifyFilter which adds filters without checking for duplicates. Invincea provided a working exploit script.

Invincea correctly identified the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.1.9.2.7 SubSpace

- **Question 41 (SC Time, Intended Not Vulnerable, Answered Yes)**

  Invincea recognized that to send a hail as a given user (impersonate a user) all that is required is a username. As a result the username is a secret which can enable a third party to impersonate a user. When registering a user, Invincea recognized that there is a time side channel in the server's response. There is a timing difference if a third party attempts to register a user that is already registered. Invincea provided a script.

  The server will respond with a registration failed if one attempts to register a username which exists in the application. With the budget of 100,000 operations, the brute force approach provided by Invincea will only work if the username is three characters or less. The provided users in the application all have usernames of 5 characters therefore this side channel does not exist.

  Post-Engagement Analysis Ruling: Incorrect

- **Question 42 (SC Space, Intended Vulnerable, Answered Yes)**

  Invincea proposed a method to utilize a side channel in space to identify a user's location. When users send a hail, the message is received by the nearest subspace user. The exploit works by creating 4 users and placing them in the centers of the SubSpace location grid quadrants. Each of the 4 exploit users can send a 'hail.' If the hail is received by another exploit user, then the corresponding quadrant is empty. This method divides the range by 4 on each step. It requires 22 steps to identify the location of the user within 1e-4 latitude and longitude. Each step requires 4 operations and it requires 8 operations to create and set the locations of the exploit users. This is a total of 96 operations, which is within the operational budget.

  Post-Engagement Analysis Ruling: Correct

A.1.9.2.8 TextCrunchr_3

- **Question 11 (AC Time, Intended Vulnerable, Answered Yes)**

  Invincea discovered that TextCrunchr 3 has a sorter which uses two sort functions. Data is first passed through the changingSort algorithm then it is sent to the quickSort function. Invincea identified that the partition points of the two merge points allow for a user to create a data set is sorted by the changingSort function into an order that is problematic for quickSort.

  Invincea correctly identified the intended vulnerability for this challenge program. The sorting algorithm has a standard O (n log (n)) behavior but certain inputs can cause it to have an O (n^2) behavior.

  Post-Engagement Analysis Ruling: Correct

A.1.9.2.9 TextCrunchr_5

- **Question 17 (AC Space, Intended Not Vulnerable, Answered Yes)**

  Invincea provided an exploit highly compressible text file which is 24 MB (less than the input guard on the TextCrunchr application) which compressed to 23 KB (less than the input budget). This text file bypasses the input guards of the program but causes the challenge program to exceed the resource usage limit.

This is an unintended vulnerability but Invincea receives credit because it is within the input budget and exceeds the resource usage limit.

Post-Engagement Analysis Ruling: Correct

- **Question 24 (AC Time, Intended Not Vulnerable, Answered Yes)**

  Invincea provided an exploit highly compressible text file which is 24 MB (less than the input guard on the TextCrunchr application) which compressed to 23 KB (less than the input budget). This text file bypasses the input guards of the program but causes the challenge program to exceed the resource usage limit.

  This is an unintended vulnerability but Invincea receives credit because it is within the input budget and exceeds the resource usage limit.

  Post-Engagement Analysis Ruling: Correct

A.1.9.2.10 TextCrunchr_6

- **Question 8 (AC Time, Intended Vulnerable, Answered Yes)**

  Invincea identified a vulnerability in the sort function. They provided an input file which uses a large set of diverse small random words. This causes the sort function to be called on a large list.

  Invincea provided an exploit which exceeded the resource usage limit and stayed within the input budget. The vulnerability within the challenge program was in cases where the length of the sort input is a multiple of 3. Invincea did not explicitly state this but the provided input file contained 100,000 words, with 99,996 unique words. This triggered the vulnerability when the sort command was called for the unique words hash table.

  Post-Engagement Analysis Ruling: Correct

A.1.9.2.11 TextCrunchr_7

- **Question 33 (AC Time, Intended Vulnerable, Answered Yes)**

  Invincea identified a vulnerability in the enigma machine part of the application. They discovered that the characters are encoded one at a time. They discovered the opportunity for a vulnerability and crafted a highly compressible file which could bypass the input program guards while staying within the input budget. The enigma machine component of the code for the large input causes the challenge program to exceed the resource usage limit.

  Invincea did not identify the intended vulnerability in the unbalanced red black tree data structure.

  Post-Engagement Analysis Ruling: Correct

A.2 Engagement 2

The overall accuracy of the Blue Teams was 77% (78% when not including the control team). Of the 175 questions answered, 109 were answered “yes” and 66 were answered “no”. The percentage of “no” responses increased significantly from 9.75% of total responses in E1 to 37.7% of total responses in Engagement 2. Because this was a “take-home” engagement, Blue Teams had more time to identify vulnerabilities and create exploits. This increase in “no” responses could be due to an increased ability to rule out vulnerabilities.

The incorrect responses by the Blue Teams for both AC and SC questions fall into four categories. These categories are not identical in both AC and SC questions but they mirror each other. For AC questions the incorrect Blue Team response categories are:

AC 21. Did not follow directions or violated the question definitions
AC 22. Failed to identify the complexity within the challenge program
AC 23. Failed to understand the input to the complexity control flow of the challenge program
AC 24. Failed to properly determine the complexity bounds of an identified complexity

For SC questions the incorrect Blue Team response categories are:

SC 21. Did not follow directions or violated the question definitions
SC 22. Failed to identify the secret and the secret location within the challenge program
SC 23. Failed to identify the side channel (processing conditioned on the secret value)
SC 24. Failed to properly analyze the side channel strength

The resulting distributions of the percentage of incorrect yes and no responses for both AC and SC questions are shown in the two tables below.

Table A-21: Engagement 2 AC Responses

	AC 1	AC 2	AC 3	AC 4
Yes	8%	13%	0%	29%
No	0%	50%	0%	0%
Total	**8%**	**63%**	**0%**	**29%**

Table A-22: Engagement 2 SC Responses

	SC 1	SC 2	SC 3	SC 4
Yes	4%	0%	15%	42%
No	0%	0%	19%	19%
Total	**4%**	**0%**	**34%**	**61%**

For AC questions, Blue Teams had the most difficulty with identifying the complexities within the challenge program and determining the complexity bounds of identified complexities. For SC questions, Blue Teams had the most difficulty with identifying the side channel and analyzing the strength of identified side channels. The tables above also show that the incorrect "No" AC responses were due to an inability to find the intended AC vulnerability. For side channels, the dominant incorrect category was reporting a side channel of insufficient strength as vulnerable. These categories (AC 2, AC 4, SC 3, and SC 4) represent the major difficulties in STAC.

A.2.1 University of Colorado Boulder

A.2.1.1 UC Boulder Overview

UC Boulder responded to 29 questions with 79% accuracy. UC Boulder responded to 12 side channel questions and 17 algorithmic complexity questions. UC Boulder's accuracy for AC time

questions was 100% and UC Boulder responded to 12 of the 16 AC time questions. UC Boulder's accuracy for AC space vulnerabilities was 40%.

UC Boulder's accuracy increased significantly from their E1 results form 30% to 79% with them answering 19 more questions. The tables below show details of UC Boulder's raw accuracy and their accuracy following the initial review.

Table A-23: Engagement 2 UC Boulder Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	4	3	75	3	75
SC Time	7	6	86	6	86
SC Space/ Time	1	1	100	0	0
AC in Space	5	2	40	2	40
AC in Time	12	12	100	12	100
Total	**29**	**24**	**83**	**23**	**79**

Table A-24: Engagement 2 UC Boulder Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	20	19	**95**	18	**90**
Not Vulnerable	9	5	**56**	5	**56**
Yes Answer	23	19	**83**	18	**78**
No Answer	6	5	**83**	5	**83**

A.2.1.2 UC Boulder Specific Responses

A.2.1.2.1 Blogger

A.2.1.2.1.1 Question 016 (AC Space, Intended Not Vulnerable, Answered Yes)

Colorado responded that there are "two main ways to interact with the Blogger web-server:

1. The user can write an (empty) entry to the blog:
   curl -X POST http://localhost:8080/Write
2. Or, the user can browse the blog:
   curl http://localhost:8080/

In order to learn whether there is a sequence of these two queries that makes the blogger server's memory consumption to go beyond 512 MB", UC Boulder "exploited a novel smart fuzzing technique based on a black-box function optimization algorithm (Covariance Matrix Adaptation Evolution Strategy CMA-ES)." Their "dynamic analysis includes four components: parameterization, string generation, CMA-ES, and fitness computation. To leverage CMA-ES, They parameterized the input space so that each input sequence can be mapped to a point belonging to multi-dimensional data in a real-vector space. More specifically," They "mapped the input space to three dimensional points, where first dimension indicates the pattern of the input 0_j1_k, 1_j0_k, $(01)_j$ and $(10)_k$ where j and k correspond to the second and third dimension

respectively, and 0 and 1 correspond to the inputs curl http://localhost:8080/ and curl –X POST http://localhost:8080/Write, respectively. Parameterization effectively transforms the search for an attack-causing input into a search for optimal values (that cause a large memory usage) for the parameters (i; j; k) in a 3-dimensional vector space. Next," UC Boulder "used CMA-ES to generate sample points based on the number of dimensions and attack sequences based on these points." They "further associated each attack sequence with a fitness (feedback) value computed based on memory usage of Blogger on this attack sequence. Based on computed fitness values for each attack sequence, the exploitr tool uses CMA-ES to learn the fittest input in order to maximize the memory usage of the Blogger server. For example, after 30 iterations" their "tool returned 11111111111111111111000000000000000000000 as an optimal input (under certain constraints put to ensure an early termination). Based on this pattern," UC Boulder "realized that posting a large number of blog entries, followed by a large number of blog accesses tend to increase the memory usage of the Blogger. This pointed" them "towards the following exploit:

```
for i in {1..250} do
        curl -X POST http://localhost:8080/Write
done
for i in {1..300} do
        curl http://localhost:8080/
done"
```

UC Boulder added that "it seems that about 300 refreshes (curl http://localhost:8080/) are required. If the PDU size includes the TCP header, then the above exploit is larger than the input budget, because the TCP header is 20 bytes. That could even mean the question could be answered by "not vulnerable", although there clearly exists a space complexity vulnerability. It might only not be exploitable given a very small input budget.

Although the size of this exploit seems to be somewhat larger than the permitted budget, it seems to capture the essence of the vulnerability […]. Blogger spawns a new thread to process a browse request that remains alive long after processing the request, contributing towards memory usage proportional to the size of the blog. Similar effects on memory performance can be achieved by posting a large single blog entry using an efficiently compressed file, followed by a large number of browse requests."

Evaluation: The challenge question provided an input budget of 5000 bytes. This represents the maximum sum of the PDU sizes of the TCP requests sent from the user to the server. For a single packet this number includes TCP Header and the TCP payload. The "curl -X POST http://localhost:8080/Write" command results in a TCP packet length of 84 bytes. This command can be called 59 times without exceeding the resource usage limit. The "curl http://localhost:8080/" command results in a TCP packet length of 78 bytes. This command can be called 64 times without exceeding the resource usage limit. UC Boulder's exploit of calling the "curl -X POST http://localhost:8080/Write" command 250 times followed by the "curl http://localhost:8080/" command 300 times exceeds the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.2.1.2.1.2 Question 040 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that, “there exists an algorithmic vulnerability in time. The LOOPR (CodeHawk) component provides a report that presents a sorted list of high complexity methods. The first three entries in the list did not lead to the discovery of any vulnerability, but inspection of the control flow graph of the fourth entry, fi.iki.elonen.URIVerifier.verify(), revealed two push() operations to a LinkedList inside a 2-deep nested loop. Inspection of the CodeHawk reverse call graph started from this function establishes that this method is reachable from the ClientHandler thread's run method. Informed by the existence of high-complexity loop from the static analysis,” UC Boulder “used CMA-ES based smart-fuzzing framework to look for input sequences with fitness function as number of iterations of the outer loop in method fi.iki.elonen.URIVerifier.verify(). A quick instrumentation of the program showed” them “that iterations of the loop are connected to the URL, and hence” they “parameterized” their “input to search for various patterns as the URLs constructed out of alphabet fa; A; 0; =g with the fitness of each string as number of executions of the outer loop in the verify method. After 26 iterations,” UC Boulder’s “tool reported the following URL:
curl http://localhost:8080/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/A/A/A/A/A/A/A/”

Evaluation: UC Boulder correctly identified the vulnerability in the fi.iki.elonen.URIVerifier.verify() method and provided a valid exploit which is less than the input budget and causes the challenge program to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.2 GabFeed 1

A.2.1.2.2.1 Question 007 (AC Time, Intended Vulnerable, Answered Yes)

Response: Vulnerable
UC Boulder responded that “there exists an algorithmic complexity vulnerability that allows the user to realize the problem to run with a cost quadratic in the size of the user input. The vulnerability occurs in the LineBreak class, which adds line breaks to a text. The code is quadratic in the number of words in the text. To exploit this most efficiently, all words should consists of one a single character. Posting a text file which consists of 200000 single character words and 200000 spaces (combined the input budget) and displaying it afterwards results in exceeding the resource usage limit. The LOOPR (CodeHawk) component's loops report lists the LineBreak.breakLines method as the most complex method in the application by both number and depth of loops.”
UC Boulder provided an exploit which posts a text file containing the pattern: “\a a a a a a a a a a a a a a a”.

Evaluation: The intended vulnerability required R&D Teams to recognize that users can set the width setting in the LineBreak.breakLines method. The method had O (n*w) asymptotic complexity where w is the width setting and n is the number of words. By setting the width to be as least as long as the number of words the user can cause the algorithm to have O (n^2) behavior. It was later realized that the width setting was not necessary to exceed the resource usage limit. UC Boulder therefore correctly identified the vulnerability within the LineBreak.breakLines method and provided a valid exploit.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.2.2 Question 010 (SC Time, Intended Vulnerable, Answered Yes)

Response: Vulnerable

UC Boulder responded that "there exists a strong side channel in time. The user controls the first argument (base) in the ModPow procedure, while the server's private key is the second argument of this procedure. Depending on whether bits in the server's private key are set to true [1], a potentially expensive computation is executed (fastMultiply). The costs of this expensive computation can be influenced by crafting the base using the publicly known third argument of the ModPow procedure."

UC Boulder stated that they identified the vulnerability using a combination of the Auditr tool and manual analysis (to adjust for the fact that the Auditr tool does not currently support reflection). UC Boulder then performed dynamic analysis by fixing the public key of a user and varying the private key. The resulting data was clustered using K-means algorithms. The dynamic analysis performed by UC Boulder allowed them to verify that the side channel was sufficiently strong.

UC Boulder provided the commands submitted to the Auditr tool and the resulting output, and the figure representing the results of the dynamic analysis performed.

Evaluation: The challenge application contained an SC Time vulnerability due to two sets of conditionals in the application. In the ModPow.modPow() method there is an if statement where the conditional is dependent on the value of each individual bit of the server's private key. Within the if statement there is a call to the OptimizedMultiplier.fastMultiply() method. This method contains an "if, else if, else" block where the conditional is dependent on the values of the variable "s", and the variable "base" which is the value submitted by an attacker. The side channel which leaks the server's private key is dependent on the differential processing times within the ModPow.modPow method and the OptimizedMultiplier.fastMultiply() method. UC Boulder identified the differential time in the ModPow.modPow() method. UC Boulder added that whether bits in the server's private key are set to 1 a potentially expensive computation is executed (fastMultiply) and that the cost of the computation can be influenced by the user input. UC Boulder's response indicates that they identified the vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.2.3 Question 038 (AC Space, Intended Not Vulnerable, Answered Yes)

UC Boulder responded that "there is an algorithmic complexity vulnerability in space. Each time a page is computed, some memory is allocated. The size of the memory allocation depends on the size of the page. The largest page that can be computed can be realized while visiting https://localhost:8080/search and searching for a space. This would produce a page of almost all public messages in the GabFeed system. However, reloading these pages requires sending about 300 bytes (mostly the cookie.txt file). Reloading the page 1300 times (to stay within the input budget) is not enough to reach the resource usage limit of 512 MB. In order to increase the recourse usage, the resulting page of https://localhost:8080/search needs to be increased. The best way to do this is by adding a message starting with a space followed by many newline characters. Each newline character increases the resulting page by 12 characters:
 .

An exploit can be created using such a file with 300000 newline characters and reloading the https://localhost:8080/search 40 times to search for a space. This can be done in about 300000+40*300 = 312000 bytes, hence less than the input budget."

Evaluation: UC Boulder identified an unintended vulnerability within the challenge program. The AC Space vulnerability described by UC Boulder can be used to create an exploit which exceeds the resource usage limit using a smaller input than the one provided in the UC Boulder exploit. The exploit provided by UC Boulder makes a large post to a GabFeed thread compose of newline characters. The exploit then reloads the search page 40 times to search for a space. The exploit provided by UC Boulder exceeded the resource usage limit while remaining within the input budget. Apogee will further investigate the impact of the JVM garbage collector on this vulnerability. While vulnerabilities within the garbage collector were out of scope for Engagement 2, UC Boulder's post-engagement analysis ruling of correct will not be changed following the investigation.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.3 GabFeed 2

A.2.1.2.3.1 Question 006 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that "there exists an algorithmic complexity vulnerability in time" in the application. "Posting a message with lots of different words which have the same hashCode () value will result in quadratic-time performance. Such a message with 10,000 words takes about 100 seconds, while such a message with 20,000 words takes about 400 seconds. The message with 20,000 words, each of 10 characters is 220,000 bytes in size, is clearly less than the input budget." UC Boulder "was able to solve this problem easily using" their "tool HashR in the default mode which computes strings with clashing HashCode values of 10 characters".

Evaluation: UC Boulder correctly identified the AC Time vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.3.2 Question 032 (SC Time, Intended Not Vulnerable, Answered No)

UC Boulder responded that there was "probably not" an SC Time vulnerability which allows an adversary to discover the server's private key. "Although there exists a minor side channel in time in the ModPow procedure, it is too weak to compute the server's private key. Based on whether bits in the private key are set to 1 or 0, a different multiplication is performed, either r0 x r0 with 0 initialized to 1, or r1 x r1 with the first value of r1 controlled by the user. The latter observation allows the user to control to some extent the loop which is based on the private key. However, the side-channel seems too weak."

Evaluation: UC Boulder correctly identified that the side channel in the ModPow method is not sufficiently strong enough to satisfy the operational budget.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.4 GabFeed 3

A.2.1.2.4.1 Question 023 (SC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that “there exists a side channel in time” within the application. “When a third party searches for a special term, the file terms text.txt is being read. This size of this file is 163982 bytes. Reading terms text.txt causes an observable delay in dealing with an HTTP request. An HTTP request that includes searching for a special term costs more than 0.3 seconds.” “On the other hand, other HTTP requests cost less than 0.3 seconds. Consequently, it is easy to distinguish between them.” UC Boulder “found this vulnerability with” their “Auditr tool, by marking the result of getDailyTerms() as secret and searching for unbalanced branches. The tool pointed to reading the terms text.txt as an unbalanced branch that depended on this secret.” UC Boulder provided the arguments which were provided to the Auditr tool in order to identify the vulnerability.

Evaluation: UC Boulder correctly identified that searches for special search terms take longer than other GabFeed user requests. UC Boulder correctly identified the SC Time vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.4.2 Question 036 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that “there is an algorithmic complexity vulnerability in time” in the application. “The method Sorter.changingSort() pushes a redundant block on the stack if the size of the list is 0 (mod 8). The LOOPR (CodeHawk) tool provides a report that lists all branch conditions. For Gabfeed-3 it reports 404 distinct branch conditions. Inspection of these branch conditions for unusual conditions produced two conditions that use the modulo operator:

(list.size() % 8)!=conditionObj0.getValue() in ccss.Sorter.changingSort

(s.length() % 2)<=conditionObj8.getValue() in ccsc.DES.parseBytes

Inspection of the control flow graph of Sorter.changingSort, […] shows that an extra basic block (starting at pc = 179) is executed if list.size() % 8 is equal to zero. Within this basic block a call is made to Sorter.changingSortHelper which pushes an extra block on the stack as shown in the call graph […]. The redundant stack block significantly increases the runtime. The user can trigger the sorting routine and can control the size of the list by adding certain messages. This vulnerability can be easily exploited within the budget, by adding 80 \hello" messages to a chat and search for \hello" after each added message. This exploit would make sure that the challenge program will exceed the resource limit regardless of the value of the modulo operation (i.e., also if the value was not 8, but smaller than 80).”

Evaluation: UC Boulder correctly identified the vulnerability within the Sorter.changingSort() method. UC Boulder correctly identified that this method was reachable and that provided a list of items with a length is divisible by 8 causes the challenge program to exceed the resource usage limit. UC Boulder provided an exploit which used the identified vulnerability to exceed the resource usage limit

Post-Engagement Analysis Ruling: Correct

A.2.1.2.4.3 Question 039 (SC Time and Space, Intended Vulnerable, Answered Yes)

UC Boulder responded that “there exists a side channel in time” in the application. “When a third party searches for a special term, the file terms text.txt is being read. The size of this file is 163982 bytes. Reading terms text.txt causes an observable delay in dealing with an HTTP request. An HTTP request that includes searching for a special term costs more than 0.3 seconds. On the other hand, other HTTP requests cost less than 0.3 seconds. Consequently, it is easy to distinguish between them. Additionally, size of the text for the various special terms differs quite a bit in most cases […]. The differences are a strong enough side channel to determine the specific 'special' term that a third party searched for.”

UC Boulder constructed “a table with average HTTP request” time for each special search term (100 samples for each term). “The worst case scenario for the provided database was when the HTTP requests cost close to 0.35 seconds. In that case, the special term could be either HIV, malaria, lupus, dyslexia, or fibromyalgia.”

UC Boulder concluded that an adversary needs “1 cost for observing the encrypted request (passive operations). That leaves […] a budget of 4 operations.” An adversary can “query the oracle for four of these terms. Either the oracle returns yes, in case the adversary knows the term. Otherwise the oracle returns four times no, suggesting that the last special term was the searched for one.”

UC Boulder “found this vulnerability with” their “Auditr tool, by marking the result of getDailyTerms() as secret and searching for unbalanced branches. The tool pointed to reading the terms text.txt as an unbalanced branch that depended on this secret.”

Evaluation: UC Boulder correctly identified the intended time side channel in the GabFeed application. They correctly concluded that this side channel was sufficiently strong to delineate searches for special search terms from other GabFeed operations. UC Boulder correctly identified that the terms_text.txt is read when a special search term is searched for. UC Boulder correctly identified that the size of the text for some special search terms differ. UC Boulder correctly identified that this difference in the size of the text for different special search terms results in a side channel which an adversary can use to determine which special search term a user searched for. UC Boulder concluded that the size difference in the special search term results created a side channel in time which could be used to uniquely identify searches for special search terms.

Given the noise the EL does not believe that the side channel identified by UC Boulder is sufficiently strong. The operational budget provides 5 operations and a probability of success of 90%. The table of mean response times to special search terms provided by UC Boulder was inconsistent with the observed response times on the reference platform (See table below).

Table A-25: GabFeed 3 Mean Response Time

Special Term	Mean UC Boulder Time (ms)	Mean Reference Platform Time (ms)	Size Special Search Term (Bytes)
Anemia	365.85	707.15	5344
Ringworm	381.72	707.23	6128
Diabetes	424.06	707.39	8912
Angina	391.95	707.51	6608
Fibromyalgia	354.22	707.52	4352
Epilepsy	404.51	707.63	7552
HIV	349.57	707.65	4048
Gonorrhea	378.15	707.68	5920
Cancer	404.72	707.7	7680
Polio	385.08	707.77	6352
Pericarditis	365.43	707.77	5216
Dyslexia	357.50	707.78	4768
Lupus	351.61	707.8	4160
Diarrhea	386.64	708.14	6560
Malaria	347.61	708.64	3904

The data collected by Apogee on the reference platform shows that by mean response time, the special search terms are separated by 0.1 ms in the worst case where 5 terms could potentially collide. Without noise, the side channel would be sufficiently strong. Accounting for noise, the range of probabilities of correctly identifying any two special search terms by response time only measured as the overlap between the response distributions is 47% in the best case to less than 1% in the worst case. We do not believe that the SC Time vulnerability discovered by UC Boulder is strong enough to satisfy the operational budget given that the adversary is only given a single time sample of a user's query. UC Boulder observed that different special search terms have responses of different sizes. UC Boulder concluded that the resulting side channel in time was sufficiently strong and did not identify the side channel in space.

Post-Engagement Analysis Ruling: Incorrect

A.2.1.2.5 GabFeed 4

A.2.1.2.5.1 Question 003 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that "there exists an algorithmic complexity vulnerability in time. Posting a message with lots of different words which have the same hashCode () value will result in quadratic-time performance. Such a message with 10,000 words takes about 100 seconds, while

such a message with 20,000 words takes about 400 seconds. The message with 20,000 words, each of 10 characters is 220,000 bytes in size, is clearly less than the input budget." UC Boulder was "able to solve this problem easily using their tool HashR in the default mode which computes strings with clashing HashCode values of 10 characters."

Evaluation: UC Boulder correctly identified the hash collision vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.6 GabFeed 5

A.2.1.2.6.1 Question 001 (SC Space, Intended Not Vulnerable, Answered Yes)

UC Boulder responded that "there exists a side channel in space. Given the encrypted packets and timings thereof within a single user session, the first step is to determine the length of the nickname of the session user. […] The first packet after logging in is the main page. This page is the same for all users except for the button to their personal page, which is their nickname." The adversary can get the length of the victim's first packet. The difference in size between the victim's first packet and the adversary's first packet reveals the difference in length between the adversary's nickname and the victim's nickname.

The next step is to compute the size of each user's public page. The adversary can compute the size in bytes of each user's public page by visiting the pages and storing the size. These pages are all the same for each user with the exception of the hyperlink to the user's personal page. Given that the adversary has the length of the victim's nickname, the adversary can determine the size of a user's public page.

The size of a user's public page (accounting to the hyperlink to the user's personal page) when the victim visits the page contains an SC Space vulnerability.

> If the victim has a private chat with a user with username: XYZ and nickname: nickXYZ, and visits the page with the user XYZ's public page, then the hyperlink "Chat with nickXYZ" is added.
>
> However, if the victim has no private chat with XYZ, then the hyperlink "Start a chat with nickXYZ" is added.

The hyperlink "also reveals the username as it points to https://localhost:8080/user/XYZ. […] Depending on whether the session user has a private chat, the size of the public messages page differs by 8 characters."

The adversary "can scan all the encrypted packets of the session user and compensate their sizes with the length of the nickname of the" victim user. "If there is a package that is exactly 8 characters smaller than a public page showing the public message of user ABC, then" the adversary "knows that user ABC is involved in a private chat with the" victim "user. There is a small probability of error, expected to be much less than 10%." The adversary "can only compute a database of sizes of all public pages (including ones for all search terms). It could be that a private chat page has exactly the size of 8 characters less than a public page showing all public messages of a user. In that case" the adversary "may think that they found such a

matching public page, while" they "actually matched a private chat page. Since" the adversary has an "accuracy on the exact size in bytes, such a double match would occur rarely. Another small probability of an error is caused by users that have no public messages. In the GabFeed 5 database there are only two such users, nacho555 and blade123, out of 30 users (thus less than 7%)."

Evaluation: There are four issues with the response provided by UC Boulder. The first issue is that the vulnerability discovered makes two assumptions not provided in the challenge question. The first assumption made is that the victim user has to navigate to the public page of user before the two can chat. The victim user can navigate to his/her own private page and resume a chat with another user without visiting that user's public page. The second assumption is that the victim cannot modify the GabFeed server (post to threads) during their session.

The second issue is that there is an assertion made by UC Boulder that the TCP payload length of an observed encrypted packet can be uniquely mapped to the size of the unencrypted HTML text within the TLS record. The GabFeed application uses a 128 bit AES cypher with Cipher block chaining. The data sent by the server is encoded in 16-byte blocks. Unless the adversary can remove the padding, it is not possible to uniquely identify the length of a victim's nickname. Apogee has been unable to determine the amount of padding added in the unencrypted TLS record header.

The adversary is allowed to observe the victim's entire user session for one cost. This leaves the adversary with one oracle query to resolve any collisions between server responses to the victim's queries. The victim can make multiple thread posts during their session. All new thread posts are returned by the server. The victim can make multiple posts during their session which collide with the packet sizes of the public profiles collected by the adversary. Given that the victim performs more than one request during his/her session, there are not enough oracle queries for the adversary to determine whether the victim's request was a visit to another user's public profile.

Post-Engagement Analysis Ruling: Incorrect

A.2.1.2.6.2 Question 004 (SC Space, Intended Not Vulnerable, Answered No)

UC Boulder responded that while "there exists a minor side channel in space in the ModPow procedure, it is too weak to compute the server's private key. Based on whether bits in the private key are set to 1 or 0, a different multiplication is performed, either r0 x r0 with 0 initialized to 1, or r1 x r1 with the first value of r1 controlled by the user. The latter observation allows the user to control to some extent the loop which is based on the private key. However, the side-channel seems too weak."

Evaluation: UC Boulder identified a potential side channel and correctly identified that the side channel was not sufficiently strong.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.6.3 Question 009 (SC Time, Intended Not Vulnerable, Answered No)

Response: Not Vulnerable

UC Boulder responded that “there is no strong side channel that a third party can identify any search term search by a user within the operational budget.” UC Boulder identified a set of 20 words that exist within the GabFeed database. UC Boulder observed the response times for these words and concluded that the observed values for the set of 20 words were not sufficiently separated to allow for a 90% probability of success.

Evaluation: UC Boulder correctly concluded that there is not an SC time vulnerability within the challenge program sufficiently strong to satisfy the operational budget.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.6.4 Question 014 (AC Time, Intended Not Vulnerable, Answered No)

UC Boulder repeated the analysis performed for the other GabFeed variants. In the case of GabFeed 5, the tools were unable to find a vulnerability. They applied the exploits identified for the other GabFeed variants; none of the exploits worked for GabFeed 5.

Evaluation: UC Boulder correctly identified that GabFeed 5 did not contain an AC time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.7 Graph Analyzer

A.2.1.2.7.1 Question 020 (AC Time, Intended Vulnerable, Answered N/A)

UC Boulder did not provide a response for this question but they provided details on why there may be a vulnerability. UC Boulder stated that their LOOPR component identified 9 high complexity (loop depth of 3 or 4) methods within the application. The “inspection of the reverse call graph” on the 9 methods showed that 3 of the 9 methods are reachable from the application’s main method. 1 of the 9 methods is not reachable. The other 5 methods are only reachable from editor commands. “There does not seem to be any command to activate the editor from the CmdProcessor provided.” The “MinimizerBarnesHut.minimizeEnergy [method] is reachable only if CmdForceLayout is put on the cmd queue in evalLayoutCmd, which requires the command-line option to be ‘force’”.

Evaluation: UC Boulder did not provide a response to this question. The vulnerability for this application is in the parsing of the input dot file. The input of a dot file references graphs by graph names and allows nodes and edges to be created for each graph. Nodes can be provided properties such as type and color. The vulnerability within the challenge program is that a subgraph can be defined in the input file. Later in the input file a node for a different subgraph can be created which reference the already defined subgraph if the node’s type contains the key word “container:” followed by the referenced subgraph’s name. A malicious user can use the allowed recursive nature of the input file to define a large graph within the input budget. This acts like a zip-bomb. In order to exceed the resource usage limit the malicious user must specify a PNG output file format. The zip-bomb-like vulnerability in the input file coupled with the PNG output format causes the resource usage limit to be exceeded.

Post-Engagement Analysis Ruling: None – no response provided

A.2.1.2.7.2 Question 034 (AC Space, Intended Vulnerable, Answered Not Vulnerable)

UC Boulder did not provide details on this conclusion. The response sheet stated that the application was "probably not" vulnerable.

Evaluation: The zip-bomb-like AC Time vulnerability in Question 020 can be used to launch an AC Space attack if the user specifies a PS (PostScript) output file instead of a PNG output file.

Post-Engagement Analysis Ruling: Incorrect

A.2.1.2.8 Image Processor

A.2.1.2.8.1 Question 005 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that "there is an algorithmic complexity. The runtime of the intensify filter depends on the RGB value of the pixels. If the value is 30 (mod 255), then the costs are significantly higher compared to other values (mod 255). UC Boulder created an exploit that consists of a 500 x 500 pixel images (25000 bytes) which only consists of the color 00001E (RBG), or 30 for only the blue component, also known as Black Russian."

Evaluation: UC Boulder correctly identified the vulnerability within the challenge program and provided a working exploit script.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.9 Law Enforcement Database

A.2.1.2.9.1 Question 026 (SC Time, Intended Vulnerable, Answered Yes)

UC Boulder identified an SC Time within the application. UC Boulder responded that "there is a vulnerability in the UDPServerHandler class that arises when a user searches for all keys in a range which allows" one "to identify the presence of a secret key. The UDPServerHandler class passes a search to the Btree which stores ID information and receives a list of all the IDs in the queried range, including restricted IDs. The UDPServerHandler iterates over this list and writes out all of the unrestricted IDs. In the event that it encounters a restricted ID, it throws and immediately catches a RestrictedAccessException, does some minor cleanup, and then returns to iterating over the list. There is a cost of approximately 3ms to passing through the try – throw – catch loop that executes when a restricted ID is found, so an attacker can use this to distinguish when the range queried contains a restricted ID."

UC Boulder provided a description of an exploit. The exploit segments the list of public keys into sets of 20. The range of each set is queried to identify the set that contain a private key. Once this set is identified the set containing the private key is segmented into subsets of 2. The subsets are queried and the gap between two public keys containing a private key is identified. Once the gap is identified a binary search in the gap can be used to identify the private key.

Evaluation: UC Boulder did not identify the intended side channel within the challenge but the one identified by UC Boulder is sufficiently strong. The application contains both a cumulative and an incremental side channel. The incremental side channel is stronger than the cumulative side channel but both are sufficiently strong. When the UDPServerHandler class receives a range query the database is queried for both public and private keys within the range. The handler

iterates through the returned list of keys within the range. If the key is public it is instantly returned in a UDP packet. If the key is private an error is thrown and handled. This results in an incremental side channel where a single query from the minimum to the maximum possible keys returns all the public keys within the database. The public keys are returned in individual UDP packets. The time difference between subsequent packets leaks information about whether a given gap between two public keys contains a private key. An adversary can run a binary search on this gap and identify the private key.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.9.2 Question 027 (SC Space, Intended Not Vulnerable, Answered No)

UC Boulder responded that the application is not vulnerable to an SC space vulnerability. UC Boulder stated that "assuming that the client cannot observe server file sizes, the opportunity for the client to observe 'space' quantities seem limited to the size of the network packets received. The only query that provides information about ranges of keys is the SEARCH query. The packet returned for the SEARCH query, however, does not allocate extra space in a packet for a restricted ID, when it occurs within a range, thus there is no evidence in the packet size for the existence of a restricted ID within the search range.
If server file sizes can be observed, however, there exists a side-channel in space that allows one to determine the value of a restricted ID in the currently provided database by observing the size of the log file distfilesys.log. For every "SEARCH" operation within two equal ranges of IDs, one range increases the number of lines (and hence the size) of the log file more if the range contains a secret ID. Based on this vulnerability, an attacker can design a binary search between the IDs 0 and 40000000 to learn the value of the secret ID within the given budget."

Evaluation: UC Boulder correctly observed that if an adversary is allowed to observe server files the size of the distfilesys.log file leaks information about the value of restricted ID's in a given search range. UC Boulder correctly recognized that the interface for an adversary to observe is limited to the packets returned by the server. UC Boulder correctly concluded that the application is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.10 SnapBuddy 1

A.2.1.2.10.1 Question 012 (SC Time, Intended Not Vulnerable, Answered Yes)

UC Boulder responded that the application contains an SC Time vulnerability. UC Boulder observed "that the time for downloading a user's image depends on the type of image [filter(s)] applied to" the image. The SnapBuddy application's 'friend invite' page contains the "name and profile photo" of each SnapBuddy user. UC Boulder sampled the download time of each user's profile photo 10 times. UC Boulder applied "K-means clustering algorithms" to the mean download time of each image and identified 6 clusters of download times. The boundaries for each cluster combined with a [normal] distribution of the download times for each image were used to define the probability of a single sample from the download time distribution of a given profile image being assigned to each cluster.
UC Boulder used a "soot-based instrumentation tool" to extract "methods […] called during the download of a user's [profile] photo." The methods called correspond to the filters applied to a given profile image. UC Boulder combined the clustering data and filter's applied data to map a

single sample profile image download time to a given profile image. The mapping function used the collected data in conjunction the CHAID algorithm to create a learned decision tree.
The resulting conclusion made by UC Boulder is that the download times of users' profile images combined with K-means clustering algorithms and the CHAID algorithm can be used to form a decision tree which can be used to map a single sample download time to the filter on a given user's profile image.
The identification of the filter(s) on a user's profile image can be used to identify the user's profile image.

Evaluation: UC Boulder reported an unintended SC Time vulnerability in the challenge program. This vulnerability hinges on the fact that the download time of each profile image coupled with information about the filters applied to a given image can be used to identify a profile image by its download time. It is currently unclear how an adversary can use the allowed observables to determine the filters applied to each image. UC Boulder extracted the methods called during the download of a user's profile image to gain this information. This does not constitute an allowed observable. Without information on how an adversary can determine the filers applied to each profile image using allowed observables the side channel identified by UC Boulder does not satisfy the operational budget using the allowed observables.

Post-Engagement Analysis Ruling: Incorrect

A.2.1.2.11 SnapBuddy 2

A.2.1.2.11.1 Question 015 (SC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that "there exists a strong side channel in time. The user controls the first argument (base) in the ModPow procedure, while the server's private key is the second argument of this procedure. Depending on whether bits in the server's private key are set to true [1], a potentially expensive computation is executed (standardMultiply). The costs of this expensive computation can be influenced by crafting the base using the publicly known third argument of the ModPow procedure." UC Boulder "found this vulnerability using the Auditr tool". UC Boulder applied the same Dynamic analysis techniques as in problem 10 (a combination of K-means algorithms and CHAIN decision tree algorithm) to verify the strength of the SC Time vulnerability.

Evaluation: The challenge application contained an SC Time vulnerability due to two sets of conditionals in the application. In the ModPow.modPow() method there is an if statement where the conditional is dependent on the value of each individual bit of the server's private key. Within the if statement there is a call to the OptimizedMultiplier.fastMultiply() method. This method contains an "if, else if, else" block where the conditional is dependent on the values of the variable "s", and the variable "base" which is the value submitted by an attacker. The side channel which leaks the server's private key is dependent on the differential processing times within the ModPow.modPow method and the OptimizedMultiplier.fastMultiply() method. UC Boulder identified the differential time in the ModPow.modPow() method. UC Boulder added that whether bits in the server's private key are set to 1, potentially expensive computation is executed (fastMultiply) and that the cost of the computation can be influenced by the user input. UC Boulder's response indicates that they identified the vulnerability in the challenge

Post-Engagement Analysis Ruling: Correct

A.2.1.2.11.2 Question 028 (SC Space, Intended Vulnerable, Answered Yes)

UC Boulder responded that "the web server is vulnerable to confidentiality leakage through side channel in space. More specifically, there is a side-channel in space based on the downloads of the profile pictures. After a user logs in, the first page downloads all the thumbnails of all the session user's friends. On the other hand, one can find the memory usage of all users by clicking on 'Send Invitation'. Matching these sizes with the download sizes of the friends after a user logs in reveals the friends identity. For example, users who apply images like 'image.png' with filters like ScaleUp will clearly be distinguishable from other users in the system. To show this," UC Boulder stated that they could "follow the exact process of the question 12 of SnapBuddy 1, but instead of time, cluster users based on their space usage. Finally," UC Boulder would "get a decision tree which shows which type of filter is applied by user's friends based on their memory usage." UC Boulder added that they "can guess the friend identity by selecting among users who apply the same type of filter."

Evaluation: The challenge question provides passive operations of observing the encrypted packets and timings thereof within a single user session. The intended SC Space vulnerability within the application was that the encrypted packet size of a user's profile image can be used to identify the user with the profile picture within the operational budget. An attacker can visit the "Send Invitation" page and get the encrypted packet size of each SnapBuddy user's profile image. By understanding the standard login process of a user an attacker can predict when the user is directed to his or her profile page after logging in. When a user is on his or her profile page, each of the user's friend's profile images are loaded. The attacker can match the size of the each encrypted profile image packet on the user's profile page to a SnapBuddy user. UC Boulder identified that the size of the profile images can be used to identify a user's friends. UC Boulder's provided a method by which an attacker can use this information to identify a user's friends. UC Boulder added that information about which filter's a given user applied could be used in conjunction with the SC Space vulnerability to identify the user. The unique size of each user's profile image allows an attacker to use the size of each encrypted profile image packet to identify the user.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.11.3 Question 035 (AC Space, Intended Not Vulnerable, Answered Yes)

UC Boulder identified an AC Space vulnerability within the application. The exploit provided by UC Boulder logs into the application, uploads a new photograph, and reapplied the same filter "filter list=F00E" 6 times. UC Boulder claims this exceeds the resource usage limit.

Evaluation: Apogee ran the exploit provided by UC Boulder several times. On one attempt a resource usage of 429 MB was observed. We did not observed the challenge application exceed the resource usage limit. The application did not contain an intended vulnerability; however, we believe that it is possible to use the exploit identified by UC Boulder to exceed the resource usage limit. We believe that this vulnerability could be due to the behavior of the garbage collector which was out of scope for Engagement 2. The AC Space vulnerability discovered by

UC Boulder receives a post-engagement analysis ruling of correct because it exceeded the resource usage limit and remained within the input budget.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.12 SnapBuddy 3

A.2.1.2.12.1 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that there is an AC Time vulnerability within the application. UC Boulder used the CodeHawk tool to perform "loop analysis and identify the filters with the highest complexity." UC Boulder sampled the parameter space using the exploitr tool to construct filter sequences that include expensive filter functions like OilFilter. UC Boulder explored the parameter space of filters and used the exploitr tool to identify sequences of filters "(OilFilter followed by ScaleUp) that contribute towards high execution time".

Evaluation: The web interface of the SnapBuddy application limits users to four filters for each image. The intended vulnerability within the application is that a malicious user can bypass this limit by making a direct POST request. UC Boulder appears to have identified this vulnerability that allowed for an exploit which applies 6 filters. UC Boulder does not appear to have recognized that there is no limit within the application on filters that can be applied through a direct POST request. The Exploit provided by UC Boulder uses more than the number of filters allowed by the web interface. The exploit provided appears to provide a witness to the vulnerability and utilize the bypass on the filter limit guard within the application. UC Boulder appears to have done more work than was required for this vulnerability in identifying high cost filters to apply. Applying any subset of the allowed filters within the input budget was sufficient to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.13 TextCrunchr 1

A.2.1.2.13.1 Question 021 (AC Time, Intended Vulnerable, Answered N/A)

UC Boulder did not officially answer this question; however in their documentation UC Boulder commented that they believe there may be a vulnerability in the ZipDecompressor method. UC Boulder stated that they had yet to construct an exploit for the vulnerability

Evaluation: The challenge application contains a zip bomb vulnerability in the ZipDecompressor method. The method will continuously unzip data from a zip archive without a depth limit. UC Boulder appears to have recognized the potential for a vulnerability in this method.

Post-Engagement Analysis Ruling: No Response given

A.2.1.2.14 TextCrunchr 2

A.2.1.2.14.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that there is an AC Time vulnerability within the application. UC Boulder used the HashR tool to identify the vulnerability. UC Boulder identified that

com.cyberpointllc.stac.hashmap.HashMap is vulnerable because long keys have the same hash code which causes hash collisions to occur.

Evaluation: UC Boulder correctly identified the intended vulnerability and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.15 TextCrunchr 3

A.2.1.2.15.1 Question 011 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that the AC vulnerability "originates in the sort method of the Sorter class. The sort method receives a list and calls two sorting functions on it, named changingSort and quicksort. ChangingSort functions like mergesort on partitions of the list larger than 2^{14} elements but just zippers the partition together on partitions of size 2^{14} or less. The quicksort implementation is naïve in that it exhibits worst case runtime when given a sorted list. ChangingSort first zippers the smallest partitions, which have size 2, and then proceeds to partitions of size 2^2, 2^3 ... up to 2^{14}." UC Boulder created "an exploit by starting with a sorted list and doing a reverse zippering, starting with partitions of size 2^{14}, and then proceeding to 2^{13}, 2^{12} ... down to 2. When TextCrunchr calls changingSort on this list the list becomes sorted so that it can be passed to quicksort where it will run for a long time."
UC Boulder identified the "vulnerability by examining the control flow graph of the mergeHelper method within the Sorter class."

Evaluation: UC Boulder correctly identified the AC Time vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.16 TextCrunchr 5

A.2.1.2.16.1 Question 017 (AC Space, Intended Not Vulnerable, Answered Yes)

UC Boulder did not provide an explanation of their response only stating that they had high confidence in the presence of a vulnerability.

Evaluation: This application does not include an intended AC Space vulnerability. UC Boulder responded that they had confidence in the presence of a vulnerability and on page 8 of their report state that there is an exploit for the identified vulnerability. The exploits folder provided by UC Boulder does not appear to contain a folder for "textcrunchr_5" where an exploit file would be expected. Until UC Boulder can provide a sufficient explanation or an exploit which provides a witness to the claimed vulnerability this response will be marked as incorrect.

Post-Engagement Analysis Ruling: Incorrect

A.2.1.2.17 TextCrunchr 6

A.2.1.2.17.1 Question 008 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder discovered an AC Time "vulnerability in the doIt1 method of the Sorter class. This vulnerability is exercised when this method tries to sort a list of words where the length of the

list is a multiple of 3. In this case the doIt1 method will call partition on the 'right' side of the list twice, causing an exponential increase in the number of calls to partition, stopping only at a base case where the lists are of size 2 or less. Furthermore, each segment that is partitioned will be completely sorted and our wealth of redundant partitioning will lead to a corresponding amount of redundant sorting.

The LOOPR (CodeHawk) component provides a report that lists all branch conditions. When" UC Boulder "generated the report, the condition with the modulo operator seemed unusual:

(this.list.size() % 3)!=0 ccss.Sorter$ClasschangingSort.doIt1

Inspection of the control flow graph of the method doIt1 […] showed that the number of pushes is not balanced (each purple node contains a push): the extra push that occurs when the list size is a multiple of 3, which is executed every time through the loop, since the length of the list does not change." UC Boulder verified the reachability of the method from the main method using the CodeHawk tool.

UC Boulder performed dynamic analysis by generating "program inputs and recording the timing behavior of the program." The input data was clustered based on the execution times using the K-means algorithm and a distance measure (Camberra Distance). There were two resulting clusters: runtime < 40 seconds and runtime > 40 seconds. A decision tree was applied using the output of their static analysis (added a num_words_mod_3) feature. Decision tree model showed that if an input has the feature of number of words % 3 = 0 and the size of the input is >5474 bytes then the input will be assigned to the >40 seconds cluster. The HashR tool was also applied in identifying the vulnerability.

Evaluation: UC Boulder correctly identified intended the vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.1.2.18 TextCrunchr 7

A.2.1.2.18.1 Question 033 (AC Time, Intended Vulnerable, Answered Yes)

UC Boulder responded that "a 250K zip file with 100,000 words takes more than 10 minutes on both TextCrunchr 2 and TextCrunchr 7, while the same file takes less than 20 secs on the other TextCrunchr variants." UC Boulder "found that TextCrunchr 7 makes use of a more sophisticated hash map but that it is vulnerable to the same type of attack as TextCrunchr 2, because it does not balance the tree when inserting values, thus degrading the tree to essentially a linked list. Profiling shows an average of 14000 loop iterations for every call to putTreeVal() in TextCrunchr 7, while TextCrunchr 1 has an average of 47 iterations per invocation of putTreeVal(). […] In putTreeVal() in TextCrunchr 1's TreeNode.java, it calls moveRootToFront() before return, while its counterpart in TextCrunchr 7 does not call moveRootToFront() before return."
UC Boulder provided a working exploit and used their HashR tool to identify the vulnerability.

Evaluation: UC Boulder correctly identified the vulnerability within the challenge program.

Post-Engagement Analysis Ruling: Correct

A.2.2 Draper Labs

A.2.2.1 Draper Labs Overview

Draper responded to 4 questions in Engagement 2 with a 50% accuracy. Draper's accuracy in this engagement was greater than their Engagement 1 accuracy however they answered 1 fewer question. Draper responded to only AC questions.

Table A-26: Engagement 2 Draper Labs Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	0	0	0	0	0
SC Time	0	0	0	0	0
SC Space/ Time	0	0	0	0	0
AC in Space	2	1	50	1	50
AC in Time	2	1	50	1	50
Total	**4**	**2**	**50**	**2**	**50**

Table A-27: Engagement 2 Draper Labs Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	2	1	**50**	1	**50**
Not Vulnerable	2	1	**50**	1	**50**
Yes Answer	2	1	**50**	1	**50**
No Answer	2	1	**50**	1	**50**

A.2.2.2 Draper Labs Specific Responses

A.2.2.2.1 Blogger

A.2.2.2.1.1 Question 016 (AC Space, Intended Not Vulnerable, Answered No)

Draper performed the same analysis for Question 16 as they did for question 40 and concluded that the application was not vulnerable.

Evaluation: Draper correctly concluded that the Blogger application did not contain an AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.2.2.1.2 Question 040 (AC Time, Intended Vulnerable, Answered No)

Draper "analyzed each method with the CAGE toolchain" because whole-program analysis was "too expensive for a program of this size. Most methods were determined to have a low complexity, with the vast majority being constant or linear. Each method was analyzed with the assumption that calls to other methods are constant time, an assumption which is unrealistic, but

gives good results in practice, as assuming that a method is constant rather than polynomial gives at most a polynomial blowup."

The first pass of the analysis excluded "most methods for excessive time complexity." Draper analyzed 258 methods. "219 were given constant or linear run-time bounds. 16 were given the answer 'MAYBE' for termination (the tool responded but could not determine termination or complexity). 23 caused a timeout for the tool (60 second timeout per method).

Of the methods for which no result was given by the tool, ~10 were examined by hand, in order of depth in the call graph starting from the main method. Several methods were determined to be non-terminating, for trivial reasons: their non-termination is intentional (or they terminate if a certain exception is thrown). These are the 'main loops' for the server, and include NanoHTTPD.start() and NanoHTTPD$ServerRunnable.run().

Other methods required careful analysis and were determined to be at worst polynomial. For example 'NanoHTTPD$HTTPSession.getBoundaryPositions(ByetBuffer, byte[])' was determined to be at worst cubic. By adding suitable annotations to the method calls and longer timeout times, the toolchain can determine this automatically.

No other methods were found to behave with excessive time resource consumption."

Evaluation: Draper did not identify the intended vulnerability within the application. The URIVerifier.verify method that processes the arguments in a provided URL is vulnerable to AC Time attacks. The method implements a regular expression parser that provided an input of repeated characters will exceed the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.2.2.2.2 Graph Analyzer

A.2.2.2.2.1 Question 013 (AC Space, Intended Not Vulnerable, Answered Yes)

Draper performed the AC Time analysis on the application before attempting the AC Space analysis. Draper did not treat the space complexity but identified that the "Acme.IntHashtable.rehash()" method shows poor space complexity if called many times in succession. Draper stated that this method is only called with exponentially decreasing frequency in the number of inserts therefore the method has a linear *amortized* complexity.

Evaluation: The application does not include any intended AC Space vulnerabilities. Draper did not provide an exploit which exceeds the resource usage limit. In evaluating this challenge Apogee did not observe an AC Space vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.2.2.2.2.2 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Draper ran their "CAGE toolchain on each method reachable from the main method, with similar caveats as described in Blogger justifications. The breakdown of the results (with 1 minute timeout for the complexity analysis) were that out of 2538 methods, 2410 were given a polynomial complexity, constant being the most common, but going up to cubic and quartic for

some methods, namely: cubic: 'org.tigris.gef.base.CmdUngroup.doIt()' and quartic: 'org.tigris.gef.base.CmdLoadDot.doIt()'. 8 methods could not be given a complexity by Draper's tools. 120 methods were too complex and the analysis timed out." Draper "hand analyzed all 8 methods and found them to all be of reasonable time usage except one, 'graph.dot.DotParser.parseDot'. This last method parses a dot file, but termination is far from obvious because of the use of 'java.io.StreamTokenizer.pushBack()' which does not allow a simple use of a simple termination argument. In fact, in this case the termination argument seems to depend on the semantic behavior of the 'graph.dot.DotParser.firstLine' and 'graph.dot.DotParser.tokenize' methods."

Evaluation: The intended vulnerability within the challenge application is that the application is vulnerable to a "billion lols" style attack. The application accepts an input of a dot file which references graphs by graph names and allows nodes and edges to be created for each graph. Nodes can be provided properties such as type and color. The vulnerability within the challenge program is that a graph can be represented by a node if node's type contains the key word "container:" followed by the referenced graph's graph name. A malicious user can provide and input file which exploits this feature to create a very large graph. This can cause the resource usage limit to be exceeded.

Draper Labs identified the graph.dot.DotParser.tokenize method as vulnerable. This method is used by the application to parse the input dot file. As part of parsing the input dot file the application recognizes the "container:" keyword used to define subgraphs therefore Draper Labs receives a post-engagement analysis ruling of correct for this response.

Post-Engagement Analysis Ruling: Correct

A.2.3 GrammaTech

A.2.3.1 GrammaTech Overview

GrammaTech responded to 14 questions with an 86% accuracy. GrammaTech attempted several side channel questions but focused primarily on algorithmic complexity questions, particularly AC Time questions. GrammaTech only answered 3 side channels questions but had a 100% accuracy for these questions.

Table A-28: Engagement 2 GrammaTech Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	1	100	1	100
SC Time	2	2	100	2	100
SC Space/ Time	0	0	0	0	0
AC in Space	3	1	33	1	33
AC in Time	8	8	100	8	100
Total	**14**	**12**	**86**	**12**	**86**

Table A-29: Engagement 2 GrammaTech Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	12	11	**92**	11	**92**
Not Vulnerable	2	1	**50**	1	**50**
Yes Answer	12	11	**92**	11	**92**
No Answer	2	1	**50**	1	**50**

A.2.3.2 GrammaTech Specific Responses

A.2.3.2.1 Blogger

A.2.3.2.1.1 Question 040 (AC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that the fi.iki.elonen.URIVerifier.verify() method was vulnerable. GrammaTech provided a trace of the method call to show that it was reachable by users. GrammaTech's IST tool identified 1 input and 8 sinks. "Through instrumentation with hprof," GrammaTech "identified the URIVerifier.verify() method as high-complexity code because it contributed significantly to the runtime on some inputs." GrammaTech manually inspected the decompiled core and confirmed the dataflow from the user input method (JavaWebServer.serve()) to URIVerifier.verify(). GrammaTech used this information to craft an input which exceeded the resource usage limit.

Evaluation: GrammaTech correctly identified the intended AC Time vulnerability in the URIVerifier.verify() method and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.2 GabFeed 1

A.2.3.2.2.1 Question 007 (AC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that the com.cyberpointllc.stac.linebreak.LineBreak.java class contained a vulnerable breakLines() method. GrammaTech provided the invocation trace of the breakLines() method. GrammaTech instrumented the application using the hprof tool and identified the "breakLines() method as high-complexity code based on the fact that it contributed significantly to the running time on some inputs." GrammaTech manually inspected the code and confirmed that the method contains doubly-nested while loops with invocations of Random.nextDouble(). GrammaTech observed that the loops do not appear to increase the time complexity of the algorithm but can delay computation. GrammaTech "observed that" the runtime of the method was "quadratic in the number of words in a given message." GrammaTech used instrumentation, call-graph reachability and manual inspection to confirm dataflow from user input to the LineBreak.breakLines() method. "Further manual analysis revealed that if a user posts a message in one server interaction and requests the same thread in a second interaction the user's message will be displayed." GrammaTech used this information to craft an exploit which exceeded the resource usage limit.

Evaluation: GrammaTech correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.2.2 Question 010 (SC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that "the secret is the server's private key. If a client that has the public key of the server wants to authenticate that server, it sends some public information to the server. The server encrypts that information with its private key and sends it back to the client. The client can use the server public key to decrypt and verify the response. The authentication is implemented in files KeyExchangeServer.java and KeyExchangeVerifier.java. The encryption function is generateMasterSecret(); this invokes ModPow.modPow(). The authentication uses the classical RSA based Diffie-Hellman algorithms. modPow() computes (public_info)^(secretKey) mod N, where serversPublicKey is the public data provided by the client, secretKey is the private key of the server, and N is the parameter of the RSA algorithm. Depending on whether the i^{th} bit is 0 or 1, the computation is different and takes a different amount of time. If the i^{th} bit is 0, s is squared, otherwise it is both multiplied and squared. The adversary can detect this timing difference by observing the wall-clock execution time of KeyExchangeVerifier.main(), which is invoked to authenticate the server before it returns a login page. This method can also be invoked on the command line using the provided script start_exchange_verifier.sh. Because the operational definitions allow the attacker to time the duration between console input and output, the timing operation falls within the scope of the engagement.
Paul C. Kocher has demonstrated that with a 128-bit key, about 700 queries are required to guarantee that the probability of guessing any correct bit is 0.95. In challenge program, the key size is 64 bits and the number of queries is proportional to the key size. Thus, an attacker require approximately 350 queries to correctly guess any bit of the private key with probability 0.95.

The implementation includes two random delays. This technique makes timing measurements inaccurate and makes the attack harder, but not impossible, as established by Kocher. The number of samples required increases roughly as the square of the timing noise. For example, if the timing characteristics have a standard deviation of 100ms, 350 timing samples are required. GrammaTech added that if it is assumed that random delay has a normal distribution with a 1-second standard deviation. Then the attack requires approximately $(350/100)^2*(1000) = 12250$ samples, well under the 60000 allowed within the question budget."

GrammaTech discovered this vulnerability by counting the number of sinks. "The method ModPow.modPow() was identified as high-complexity code as it contains 5 sinks. Numerous other methods related to encryption were also identified as high-complexity based on high sink count, notably in the classes DES and OptimizedMultiplier. Manual inspection of the code confirmed that the function modPow contains an if-statement whose branches have different execution time." GrammaTech devised an attack manually based on domain knowledge about RSA algorithms and the references mentioned above.

Evaluation: The challenge application contained an SC Time vulnerability due to two sets of conditionals in the application. In the ModPow.modPow() method there is an if statement where the conditional is dependent on the value of each individual bit of the server's private key. Within the if statement there is a call to the OptimizedMultiplier.fastMultiply() method. This

method contains an “if, else if, else” block where the conditional is dependent on the values of the variable “s”, and the variable “base” which is the value submitted by an attacker. The side channel which leaks the server’s private key is dependent on the differential processing times within the ModPow.modPow method and the OptimizedMultiplier.fastMultiply() method. GrammaTech’s response indicates that they found the vulnerability in the ModPow.modPow() method. In a later section of GrammaTech’s response they stated that the application takes a different amount of time whether the i^{th} bit is 0 or 1. The details on the SC Time vulnerability provided by GrammaTech identify a side channel which leaks the Hamming weight of the server’s private key; this side channel is not strong enough to satisfy the operational budget. GrammaTech referenced a paper by Paul C. Kocher which detailed an exploit which can leak the server’s private key. If GrammaTech had implemented the exploit in the Paul C. Kocher paper, it would have worked. GrammaTech therefore receives a post-engagement analysis ruling of correct for this response.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.3 GabFeed 2

A.2.3.2.3.1 Question 006 (AC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that the HashMap implementation in “com.cyberpointllc.stac.hashmap, in particular methods TreeNode.find() and TreeNode.putTreeVal()” were vulnerable. GrammaTech provided the invocation traces for the two methods. GrammaTech used their IST tools to identify 6 inputs and 76 sinks. GrammaTech “prioritized methods with a high number of complexity sinks” and “identified several methods in the TreeNode class and high-complexity code.” GrammaTech focused on the TreeNode.find() and TreeNode.putTreeVal() methods. GrammaTech checked the reachability call-graph, examined the results of the hprof instrumentation and used manual inspection of decompiled code to verify data flow to TreeNode.find() and TreeNode.putTreeVal() from a number of inputs. “Manual inspection of the TreeNode methods revealed they are part of a HashMap implementation. Large buckets are implemented as trees. Within the trees, entries are ordered by a hash of the key (computed using its Java hashCode()), and if two keys have the same hash, by the key itself. The implementation does not appear to contain logic that would keep the trees balanced.

This led to the intuition that a worst-case attack would insert numerous entries with the same hashCode() to create large buckets and large trees. This would cause the trees to degenerate to linked lists and lead to a quadratic running time for posting a message (as a function of the number of words in the message). Based on the HashMap intuition and the identified data flow from NewMessageHandler, where the HashMap keys are strings,” GrammaTech “crafted an attack with a large number of strings having the same hashCode().”

Evaluation: GrammaTech correctly identified the intended AC Time vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.4 GabFeed 3

A.2.3.2.4.1 Question 023 (SC Time, Intended Vulnerable, Answered N/A)

GrammaTech responded that "when a user searches for a term, the response time depends on whether the term is a 'special' term. This is seen in SearchHandler.handlePost(), where additional computation is executed if the term is 'special'. […] There are known attacks in the literature where attackers observe HTTP request/response pairs to infer counts of values based on response time. […] Because the challenge question allows the attacker to observe the encrypted request and corresponding response packets and timings thereof for a single user session," GrammaTech "believes an attack may be possible." However, GrammaTech was "unable to confirm that this attack is feasible within the challenge question budget."

Evaluation: While GrammaTech did not officially respond to the question, they identified the intended SC Time vulnerability in the challenge application. Given that the number of special terms is limited and publicly available, GrammaTech could have confirmed the strength of this vulnerability by sampling the special search term 'search' operation and compared the times with samples of other GabFeed interactions.

Post-Engagement Analysis Ruling: No ruling to be made as no response was provided.

A.2.3.2.5 GabFeed 4

A.2.3.2.5.1 Question 003 (AC Time, Intended Vulnerable, Answered Yes)

GrammaTech repeated the exploit used in GabFeed 2 Question 006 to exceed the resource usage limit.

Evaluation: The exploit provided by GrammaTech in GabFeed 2 Question 006 used an unbalanced tree vulnerability to exceed the resource usage limit. The vulnerability in this variant of GabFeed is that it uses a hash table based on a red black tree. On 'put' calls, the tree is not balanced. An attacker can exceed the resource usage limit by posting a large message containing words with hash collisions. The exploit identified by GrammaTech in GabFeed 2 is sufficient to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.6 Graph Analyzer

A.2.3.2.6.1 Question 034 (AC Space, Intended Vulnerable, Answered No)

GrammaTech responded that "No vulnerability was found with" their tools. Detailed manual analysis of the decompiled code lead Gramma to believe no vulnerability exists. GrammaTech's IST tool "identified 1 input and 104 sinks." "There were no sinks with high complexity related to creating the output image. Domain knowledge led to the idea that" GrammaTech "could force the program to output a large PNG file by passing in an input crafted to reduce the effectiveness of PNG compression. Manual inspection of the decompiled code showed that all parameters relevant to reducing the effectiveness of PNG compression are fixed and independent of the input. This includes the height and width of the image, the edge colors, and the size of the nodes. This fact, combined with the 5000 byte limit on the input file, appears to make it impossible to force the program to create a large postscript or PNG file."

Evaluation: The intended vulnerability within the challenge application is that the application is vulnerable to a “billion lols” style attack. The application accepts an input of a dot file which references graphs by graph names and allows nodes and edges to be created for each graph. Nodes can be provided properties such as type and color. The vulnerability within the challenge program is that a graph can be represented by a node if node’s type contains the key word “container:” followed by the referenced graph’s graph name. A malicious user can provide and input file which exploits this feature to create a very large graph. GrammaTech correctly observed that the PNG output format was limited in its output size; the postscript output format however is not. A malicious user can use the “container:” keyword to define nodes that allow predefined graphs to be referenced as subgraphs. This vulnerability in conjunction with the postscript output format can be used to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.2.3.2.7 Image Processor

A.2.3.2.7.1 Question 005 (AC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that com.stac.mathematics.exp() was vulnerable. GrammaTech’s IST tool identified 13 inputs and 36 sinks in the challenge application. GrammaTech “identified Mathematics.exp() as a high-complexity method based on high sink count. hprof instrumentation confirmed high cost and high invocation counts for Mathematics.exp() and Mathematics.factorial() (invoked by exp()). By checking call-graph reachability, examining the results of hprof instrumentation, and manual inspection of decompiled code,” GrammaTech “verified data flows to Mathematics.exp() from Main.clusterAgainsTrainingSet(). Manual inspection of Mathematics.java revealed the loop in exp is executed between 4 and 1003 times. Mathematics.exp() itself is invoked inside a doubly-nested loop that iterates over all the pixels of an image, in Intensify.filter(). In the worst case, the exp() loop could run 250,750,000 times. The value that determines the number of iterations in Mathematics.exp is retrieved from the static array in Mathematics using the RGB values of the input image's pixels. Manual inspection of the code revealed that the maximum possible value of 1003 is achieved when the red value is 0 and the sum of RBG values is 285.” GrammaTech used this information to craft an input which exceeds the resource usage limit.

Evaluation: GrammaTech correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.7.2 Question 030 (AC Space, Intended Not Vulnerable, Answered No)

GrammaTech responded that “no space vulnerability was found with their tools. Detailed manual analysis of the decompiled code leads them to believe no vulnerability exists.” GrammaTech’s tools found no space vulnerabilities. “Visual inspection showed that most containers have constant or linear size in the number of images, or are likely to be garbage collected.” GrammaTech “did not identify any opportunity to force the creation of a large number of objects.” GrammaTech concluded that “no space vulnerabilities are apparent.”

Evaluation: GrammaTech correctly concluded that the challenge application did not contain an intended AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.8 Law Enforcement Database

A.2.3.2.8.1 Question 026 (SC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that "clients of the law enforcement database can issue queries that return all unrestricted employee IDs within a specified range. If the specified range also includes some restricted IDs, which are the secret data for this question, those will not be returned in the result set. However, the presence of restricted IDs in the search range will make the query run slightly slower, for two reasons. First, every ID in the range, whether restricted or unrestricted, must be processed by the query-processing code, and that takes time. Second, if no restricted ID is encountered during a search, the log file lastaccess.log is written only twice, but if a restricted ID is encountered during the search, lastaccess.log is written at least three times, and this also causes the search to be slightly slower." GrammaTech proposed an exploit in which an attacker can binary search the range of possible keys and identify the restricted ID in the provided database.

GrammaTech used their IST tool to rank methods by number of sinks. The highest-ranking four methods identified were in the B-tree implementation. GrammaTech used this information in addition to manual analysis to identify the SC Time vulnerability.

Evaluation: The challenge application includes both a differential and a cumulative side channel. When a user submits a query the application calls the B-tree database and is provided a list of restricted and non-restricted IDs in the search range. The application iterates through the list of IDs; if an ID is not restricted it is returned in a single UDP packet; if an ID is restricted the application throws and handles an error and does not return the ID. When the user receives the list of public IDs the time between the first and last packet is a cumulative SC in time; the time between subsequent packets is a differential SC in time. With a single query an attacker can use the differential SC time vulnerability to identify a gap between non-restricted IDs where a restricted ID lies. The attacker can then binary search the gap and identify the value of the restricted ID. The differential side channel was the intended vulnerability; the cumulative side channel was not intended to be strong enough to reveal the value of a restricted ID. The cumulative side channel contains too much noise for the binary search exploit proposed by GrammaTech. There is an exploit which uses the cumulative side channel to reveal the value of a restricted ID within the operational budget. The cumulative side channel is strong enough that an attacker can submit a single query of two subsequent non-restricted IDs and determine whether a restricted ID lies in the gap between the two subsequent non-restricted IDs. The attacker has enough operations to individually query each gap in the provided database of 330 non-restricted IDs. The attacker can identify the gap containing the restricted ID and binary search it to identify the value of the restricted ID. While the exploit provided by GrammaTech does not satisfy the operational budget, R&D Teams are not required to provide a working exploit. Because there exists an exploit using side channel discovered by GrammaTech which satisfies the operational budget, GrammaTech receives a post-engagement analysis ruling of correct for this response.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.9 SubSpace

A.2.3.2.9.1 Question 042 (SC Space, Intended Vulnerable, Answered Yes)

GrammaTech discovered by manual analysis that the challenge application uses a "GeoMultiTrie data structure which divides the location space into a tree of nested rectangle. The root rectangle represents the entire Earth, and every non-root rectangle is a quadrant of its parent, so the structure is essentially a quadtree. Furthermore, a non-empty GeoMultiTrie always has a fixed height. User locations are only stored at the leaf level, where rectangles have a width of less than 10^-4 degrees. Every user location in the GeoMultiTrie must be located within one rectangle at each depth level of the tree. If two users' locations belong in the same rectangle at a particular depth level, the rectangle at that level is only allocated (and stored on disk) once. The sharing of the rectangle object creates a difference in file size when the database is stored on disk. The maximal level of rectangle sharing is achieved only when two users are both located in the same leaf-level rectangle, which has width less than 10^-4 degrees." GrammaTech discovered that the database is backed by a database file. An attacker can observe changes in the file size and use this in addition to triangulation to identify the location of a user. GrammaTech noted that this side channel assumes that an attacker can observe the size of the Subspace database file. GrammaTech added that "there is an inconsistency/ambiguity" in the question because the allowed operations "does not specifically list file-size observables among the permitted operation."

Evaluation: GrammaTech identified an SC Space vulnerability within the challenge application. The vulnerability discovered by GrammaTech was the initial intended vulnerability within the challenge application. It was discovered in review that it is possible to triangulate a user's location by using the anonymous communication protocol of the challenge program. An attacker can create two users and move the users about the grid. If a message sent from one user arrives at the other then there are no other users closer to either user. An attacker can use this feature to triangulate the location of a user. GrammaTech correctly observed that the challenge question does not provide any passive operations to allow for the observation of the Subspace database file. Because the vulnerability discovered by GrammaTech was the original intended vulnerability and GrammaTech correctly observed that the required observable was not provided in the challenge question, GrammaTech receives a post-engagement analysis ruling of correct for this response.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.10 TextCrunchr 2

A.2.3.2.10.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that "the sorting code (com.cyberpointllc.stac.sort.Sorter) is vulnerable because the input affects the number of times support methods are called. In particular, the 'changingSort' and 'merge' methods in that class are executed more frequently with larger inputs of a certain type (described below) and subsequently lead to high execution times." GrammaTech used their IST tool to identify the word frequency calculations and subsequent sorting of the data as vulnerable. GrammaTech "manually examined the sorting routines in Sorter.java and inferred the type of sort (i.e., merge, as mentioned above) and devised a likely exploit based on the knowledge of the contents (strings). With the goal of creating many unique strings within the compressed-size input budget," GrammaTech created an exploit to confirm the

vulnerability. GrammaTech's exploit used "unique strings" which "hash to the same value to ensure that hash codes cannot be used for faster string inequality tests" and noted that this "may or may not be important". GrammaTech uses the same strings in this exploit as the ones used for the GabFeed 2 AC Time vulnerability. GrammaTech added that the implemented sort is a "merge-type sort with O (n log (n)) worst-case".

Evaluation: The intended vulnerability in the challenge application was in the Hash Table implementation. The Hash Table is vulnerable to hash collisions. A compressed file input with enough word collisions can exceed the resource usage limit. The response provided by GrammaTech indicates that they suspected that the sorting code was vulnerable however in crafting a vulnerable input GrammaTech noted that they used unique string which hash to the same value. This indicates that GrammaTech may have noted the vulnerable Hash Table implementation. GrammaTech provided an exploit description which can be used to create a valid exploit to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.11 TextCrunchr 5

A.2.3.2.11.1 Question 017 (AC Space, Intended Not Vulnerable, Answered Yes)

GrammaTech responded that in the TextCrunchr application," input is read and processed before Sorter is called to organize the output (with respect to word frequency). The sorting routine calls a ""changingSort"" method which, in this version of TextCrunchr, has been extracted into its own class. As such, every call to this method creates a new Classchanging sort object, taking up more memory. This could likely be exploited by either making the sort harder (i.e., more iterations of fundamental sub-steps) or providing more data that needs to be sorted to overrun the 512mb limit for this problem." GrammaTech observed that "certain inputs cause the number of times ClasschangingSort.doIt1 is called to increase drastically. It appears that the ordering of the data gives less signal (if any at all) than the sheer number of elements to be sorted. As such," GrammaTech "posit that such an attack would contain many unique words, but be readily compressible to fit within the input budget (400,000 bytes)". GrammaTech used their IST tool to identify the vulnerability and confirm the reachability of the vulnerable code.
GrammaTech tested various inputs and managed to use 280 MB of RSS memory. GrammaTech "did not explore the tradeoffs between string length, uniqueness, and ease of comparison sufficiently." GrammaTech responded that the application was vulnerable but noted that they had "low-confidence" in this response.

Evaluation: The challenge application did not contain an intended AC Space vulnerability. GrammaTech claimed to have identified a vulnerability but did not exceed the resource usage limit. Apogee tested various inputs and was unable to confirm an unintended vulnerability. The AC Space vulnerability identified by GrammaTech cannot be used to exceed the resource usage limit given the input budget.

Post-Engagement Analysis Ruling: Incorrect

A.2.3.2.12 TextCrunchr 6

A.2.3.2.12.1 Question 008 (AC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that ”this version of TextCrunchr is vulnerable in the same way as TextCrunchr 2, with most steps and conclusions holding as they did for TextCrunchr 2. GrammaTech noted that de-compilation examination showed implementation changes, such as the extraction of the “changingSort” method to an inner class. This did not appear to have an important effect on the problem.”

Evaluation: GrammaTech’s initial response did not explicitly identify the intended AC Time vulnerability in the challenge application. This variant of TextCrunchr has a sorting algorithm which normally has O (n log n) behavior. Provided an input list which has length divisible by 3, the behavior of the sort becomes O (n^2). GrammaTech noted in their response to question 37 in TextCrunchr 2 that “the implemented sort is a merge-sort [which has an] n * log (n) worst-case.” The O (n log n) behavior of the sorting algorithm in TextCrunchr 6 is not vulnerable. The exploit description provided by GrammaTech for TextCrunchr 2 works because the application contained a hash table vulnerability and was vulnerable to inputs containing hash collisions. This variant of TextCrunchr does not contain this vulnerability. After the initial Engagement 2 analysis, GrammaTech provided the exploit which was used to demonstrate the vulnerability. GrammaTech added that their exploit exceeded the resource usage limit and was created during Engagement 2. GrammaTech’s exploit was run by Apogee and exceeded the resource usage limit. GrammaTech’s exploit worked because used exactly 30,000 unique word. This feature of the exploit triggered the intended vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.3.2.13 TextCrunchr 7

A.2.3.2.13.1 Question 033 (AC Time, Intended Vulnerable, Answered Yes)

GrammaTech responded that “this version of TextCrunchr is susceptible to the same attack as versions 2 and 6. Inspection of the decompiled code showed that the “changingSortHelper” method introduced 7 partitions (yielding 8 sections) rather than 1. While this increases complexity, it also has the potential to increase efficiency. Nonetheless, the same attack as in cases 2 & 6 demonstrated a vulnerability.”

Evaluation: The challenge application uses a red black tree-based hash table which is vulnerable to hash collisions. In input file with enough collisions can cause the resource usage limit to be exceeded. This challenge application is vulnerable to hash collisions as TextCrunchr 2 was. As stated above TextCrunchr 6 does not contain a hash collision vulnerability. The exploit provided for TextCrunchr 2 will work for this challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.4 Iowa State University

A.2.4.1 Iowa State Overview

Iowa State responded to 13 questions with a 69% accuracy. Iowa State answered 7 side channel questions and 6 complexity questions.

Table A-30: Engagement 2 ISU Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	3	3	100	3	100
SC Time	3	2	67	1	33
SC Space/ Time	1	1	100	1	100
AC in Space	1	0	0	0	0
AC in Time	5	4	80	4	80
Total	**13**	**10**	**77**	**9**	**69**

Table A-31: Engagement 2 ISU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	10	10	**100**	9	**90**
Not Vulnerable	3	0	**0**	0	**0**
Yes Answer	13	10	**77**	9	**69**
No Answer	0	0	**0**	0	**0**

For GabFeed 5 question 001, Iowa State identified a vulnerability which allows an attacker with a GabFeed account to view any chat in the GabFeed application. This was an unintended vulnerability in the application and did not quite fit the STAC definition of an SC Space vulnerability. The response by Iowa State received a post-engagement analysis ruling of correct because it was a vulnerability in the challenge application. This response coupled with a weak side channel identified in question 026 resulted in the change in the number correct following a review of the responses provided by Iowa State.

A.2.4.2 Iowa State Specific Responses

A.2.4.2.1 Blogger

A.2.4.2.1.1 Question 016 (AC Space, Intended Not Vulnerable, Answered Yes)

Iowa State responded that using their "call graph smart views," they "identified that when the app is started, the server listener thread ServerRunnable is started, and each client request is processed in a separate thread of type ClientHandler." Iowa State's "Loop Call Graph (LCG) tool was used to identify 15 methods with loops that were reachable from ClientHandler.run, one or more of which may cause a vulnerability. Based on this, Iowa State performed dynamic analysis on those 15 methods, one of which actually contained the vulnerability, namely URIVerifier.verify(). URIVerifier.verify() method contains 3 loops, two loops nested within an outer loop. The outer loop pops and processes elements from a stack until it is empty. The two nested loops push into the stack. The data structure URIElement is recursive, i.e., contains a list that contains URIElements." Iowa State "hypothesized that due to the recursive data structure, URIVerifier.verify may be vulnerable to AC in Space attack. Dynamic analysis revealed that the size of the stack iterated in the URIVerifier.verify method increased and decreased unpredictably, and eventually exceeded the resource usage limit of 512 MB to process a large URL string within the budget (5000 bytes). URIVerifier.verify() is triggered whenever the server

receives a HTTP request. Particularly, the URL string passed to the server is sent as parameter to URIVerifier.verify().”

Evaluation: The challenge application did not include an intentional AC Space vulnerability. Apogee attempted to recreate the unintended AC Space vulnerability reported by Iowa State but was unable to exceed the resource usage limit. Without a working exploit this response receives a post-engagement analysis ruling of incorrect.

Post-Engagement Analysis Ruling: Incorrect

A.2.4.2.1.2 Question 040 (AC Time, Intended Vulnerable, Answered Yes)

Iowa State responded that using their “call graph smart views”, they “identified that when the app is started, the server listener thread ServerRunnable is started, and each client request is processed in a separate thread of type ClientHandler.” Iowa State’s “Loop Call Graph (LCG) tool was used to identify 15 methods with loops that were reachable from ClientHandler.run, one or more of which may cause a vulnerability.” Based on this, Iowa State “performed dynamic analysis on those 15 methods, one of which actually contained the vulnerability, namely URIVerifier.verify(). URIVerifier.verify() method contains 3 loops, two loops nested within an outer loop. The outer loop pops and processes elements from a stack until it is empty. The two nested loops push into the stack. The data structure URIElement is recursive, i.e., contains a list that contains URIElements.” Iowa State “hypothesized that URIVerifier.verify may be vulnerable due to the above reasons. Dynamic analysis revealed that URIVerifier.verify consumed increasing amount of time to process URLs of increasing lengths. Eventually, the time consumed exceeded the resource usage limit of 300 seconds to process a URL string containing 35 ‘a’s. URIVerifier.verify() is triggered whenever the server receives a HTTP request. Particularly, the URL string passed to the server is sent as parameter to URIVerifier.verify().”

Evaluation: Iowa State correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.4.2.2 GabFeed 3

A.2.4.2.2.1 Question 023 (SC Time, Intended Vulnerable, Answered Yes)

Iowa State responded that “search results for both special and non-special search terms are formatted using LineBreak.breakLines before being sent to the client, so search time = search time + formatting time. Three loops in LineBreak.breakLines were identified to be non-monotonic, i.e., contained statements that modified variables influencing their termination condition in different directions. This led” Iowa State “to hypothesize that size of the messages posted in the GabFeed database could induce timing differences for search results. “

The challenge application “includes a text file containing results for special search terms, and stores user posted message in memory in the GabFeed database. SearchHandler.getInfoText retrieved contents from the text file and SearchHandler.getContents retrieved contents from the in memory GabFeed database. Results for a special search term are retrieved from the GabFeed database and the text file, while the results for non-special search terms are retrieved only from the GabFeed database.” Iowa State was “able to construct an inter-procedural EFG with the above methods as the input. This EFG informed them that any special search term takes time

proportional to the sum of (a) the sizes of the messages in the GabFeed database containing that special search term, and (b) the size of the message in the text file containing that search term. Search of a non-special search term takes time proportional to the sum of sizes of the messages containing that term in the GabFeed database. The above difference in searching of special and non-special search terms constitutes a timing side channel."

Iowa State proposed an exploit where an attacker populates the GabFeed database with special search terms. "This will cause the search of any special term by any user to take significantly more time than search for non-special terms (because the posted large message will take longer to process)." Iowa State added that "this timing difference was sufficiently large on the reference platform, so that an attacker can tell whether the particular operation was a search operation using a special search term or not by passively observing the times taken for that operation by the user."

Evaluation: Iowa State identified the intended SC Time vulnerability in the application. On the reference platform with the database provided, the time to retrieve the special search term information from a text file should be large enough to allow an attacker to distinguish between searches for special search terms and other GabFeed operations. An attack should not require an attacker to amplify the separation between searches for special search terms and other GabFeed operations.

Post-Engagement Analysis Ruling: Correct

A.2.4.2.2.2 Question 039 (SC Time and Space, Intended Vulnerable, Answered Yes)

Iowa State responded that "search results for both special and non-special search terms are formatted using LineBreak.breakLines before being sent to the client, so search time = search time + formatting time. Three loops in LineBreak.breakLines were identified to be non-monotonic, i.e., contained statements that modified variables influencing their termination condition in different directions. This led" Iowa State "to hypothesize that size of the messages posted in the GabFeed database could induce timing differences for search results. "

The challenge application "includes a text file containing results for special search terms, and stores user posted message in memory in the GabFeed database. SearchHandler.getInfoText retrieved contents from the text file and SearchHandler.getContents retrieved contents from the in memory GabFeed database. Results for a special search term are retrieved from the GabFeed database and the text file, while the results for non-special search terms are retrieved only from the GabFeed database." Iowa State was "able to construct an inter-procedural EFG with the above methods as the input. This EFG informed them that any special search term takes time proportional to the sum of (a) the sizes of the messages in the GabFeed database containing that special search term, and (b) the size of the message in the text file containing that search term. Search of a non-special search term takes time proportional to the sum of sizes of the messages containing that term in the GabFeed database. The above difference in searching of special and non-special search terms constitutes a timing side channel. The size of the response from the server containing the search results is also a side channel in space, as messages with different sizes can be identified with corresponding search terms."

Iowa State proposed an exploit where an attacker populates the GabFeed database with special search terms. This will cause the search of any special term by any user to take longer than

search for non-special terms (because the posted large message will take longer to process). Iowa State added that this timing difference was sufficiently large on the reference platform, so that an attacker can tell whether the particular operation was a search operation using a special search term or not by "passively observing the times taken" for that operation by the user. Once a user operation is identified as a search for a special search term Iowa States exploit uses the SC Space vulnerability to map the size of the server response to a special search term.

Evaluation: Iowa State correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.4.2.3 GabFeed 5

A.2.4.2.3.1 Question 001 (SC Space, Intended Not Vulnerable, Answered Yes)

Iowa State "constructed an inter-procedural EFG with the methods GabDatabase.getChat() and GabChat.getUserIDs() as input. The inter-procedural EFG revealed paths from GabHandler, ChatHandler and UserHandler that called these methods. In all except one path, GabChat.getUserIDs was invoked before GabDatabase.getChat was invoked. The path where GabDatabase.getChat was called without requiring GabChat.UserIDs to be called led Iowa State to hypothesize that this path could cause the SC vulnerability. Further examination of the path revealed that it was triggerable only when a new chat had been created." Iowa State provided an exploit where an attacker uses this vulnerability to view other user's chats.

Evaluation: The unintended vulnerability discovered by Iowa State is a side channel. This unintended side channel does not quite fit the STAC definition of a space side channel. This vulnerability is present within the challenge application therefore Iowa State receives a post-engagement analysis ruling of correct for this response.

Post-Engagement Analysis Ruling: Correct

A.2.4.2.4 Law Enforcement Database

A.2.4.2.4.1 Question 019 (AC Time, Intended Not Vulnerable, Answered Yes)

Iowa State responded that "using PUTRECORD," an attacker "can put a file on the server and receive that file using GETRECORD." If an attacker performs a "GETRECORD [request] on a restricted ID, then the client waits indefinitely; thus exceeding the resource usage. This happens in UDPServerHandler.java, case 13 of the channelRead0 method.To trigger the vulnerability use the PUTRECORD command, and give record name as 'ID:' where ID is a restricted ID in the database. Give a random string for record contents. A file named <ID>: will be created on the server. Retrieve that file using GETRECORD. The client will wait indefinitely, thus exceeding the resource usage." Iowa State used their tools and "an EFG with an event based on the restricted ID, to find all paths in the channelRead0 method where a restricted ID was used. Dynamic analysis on one such path revealed the vulnerability."

Evaluation: The challenge application description document restricted the user interactions to INSERT and SEARCH operations. The PUTRECORD and GETRECORD operations are not valid operation for this challenge. The application did not contain an intended AC Time vulnerability within the scope of the description-defined user interactions.

Post-Engagement Analysis Ruling: Incorrect

A.2.4.2.4.2 Question 026 (SC Time, Intended Vulnerable, Answered Yes)

Iowa State responded that "INSERT operation takes different times to insert a new key into database based on when it is being inserted. Using SEARCH attacker can get all the non-private keys corresponding to unrestricted IDs. Based on this and with knowledge of special property of BTree data structure (which causes side channel)" an attacker "can insert new keys and predict behavior of the side channel (which insert will take more time)." The attacker can use this behavior to narrow down the range of IDs in which a restricted ID lies. "Eventually the range gets small enough for the attacker to brute force the secret (a private key corresponding to a restricted ID) using oracle queries."

Evaluation: The side channel discovered by Iowa State was the initial vulnerability intended for the application. During review it was discovered that the application contained two much stronger side channels. The operational budget was set to exclude the side channel discovered by Iowa State. The intended vulnerability is that the application includes both a differential and a cumulative side channel. When a user submits a query the application calls the B-tree database and is provided a list of restricted and non-restricted IDs in the search range. The application iterates through the list of IDs; if an ID is not restricted it is returned in a single UDP packet; if an ID is restricted the application throws and handles an error and does not return the ID. When the user receives the list of public IDs the time between the first and last packet is a cumulative SC in time; the time between subsequent packets is a differential SC in time. With a single query an attacker can use the differential SC time vulnerability to identify a gap between non-restricted IDs where a restricted ID lies. The attacker can then binary search the gap and identify the value of the restricted ID. The differential side channel was the intended vulnerability; however the cumulative side channel is sufficiently strong given the operational budget. With the cumulative side channel an attacker can submit a single query of two subsequent non-restricted IDs and determine whether a restricted ID lies in the gap between the two subsequent non-restricted IDs. The attacker has enough operations to individually query each gap in the provided database of 330 non-restricted IDs. The attacker can identify the gap containing the restricted ID and binary search it to identify the value of the restricted ID. The vulnerability discovered by Iowa State is not sufficiently strong to satisfy the operational budget.

Post-Engagement Analysis Ruling: Incorrect

A.2.4.2.5 SnapBuddy 1

A.2.4.2.5.1 Question 012 (SC Time, Intended Not Vulnerable, Answered Yes)

Iowa State responded that "the amount of time it takes for a profile picture to load is suspected to be unique for each person. When a user logs in, he can record the time it takes to load picture of each user by going to the Send Invitation page. The Send Invitation page loads the picture of each user. Once that is done, an attacker can log out and look at the network for request/responses. When a user logs in, it shows all his friends. By looking at the time it takes to load images, the attacker can figure out the user's friends. The reason a profile picture takes different times to load is because of the ability to add "filters" to a profile photo. You can apply up to 4 filters to a picture. Because of this, the time taken to load the profile picture can vary depending on which filters are chosen." Their tool "flagged some loops in filters as expensive

loops. This was helpful for" Iowa State "to hypothesize that applying different filters would cause difference in timing and be a potential side channel where the filters were used."

Evaluation: Iowa State reported an unintended SC Time vulnerability. The side channel discovered by Iowa State allows an attacker to map the download time of a profile image to the profile image itself and by extension the user with the profile image. The EL does not believe that this side channel satisfies the operational budget using the allowed observables.

Post-Engagement Analysis Ruling: Incorrect

A.2.4.2.5.2 Question 031 (SC Space, Intended Vulnerable, Answered Yes)

Iowa State responded that "when a user has to change the location or set the initial location, the user has to provide the entire list of BSSIDs corresponding to that city (location). The attacker can get a 'csv' file of list of city to corresponding string of BSSIDs by going to "/cities" URL and downloading the csv file. The length of this string is unique for each location as can be seen from the csv file. The attacker can deduce the location of a user by passively observing the length of the location string contained in the packets sent to the server." Iowa State's "tool was helpful in tracing the flow of location information in the code."

Evaluation: Iowa State correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.4.2.6 SnapBuddy 3

A.2.4.2.6.1 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

Iowa State responded that an attacker can "log in to the default account and select a profile picture." The attacker "can then select all the 6 filters as "Oil Filters" and click on Apply. Doing this a couple of times exceeds the budget on the reference platform by staying within the input budget." Iowa State's "tool flagged a loop in the "OilFilter" class as a potentially expensive loop. Iowa State was able to trigger this loop by selecting the "OilFilter" as the filter."

Evaluation: Iowa State correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.4.2.7 SubSpace

A.2.4.2.7.1 Question 042 (SC Space, Intended Vulnerable, Answered Yes)

Iowa State responded that an "attacker starts off by registering two malicious users (drone 1 and 2) on the server and starts them off at the same location (this location is randomly chosen). Then he moves one of the ships along the one axis (i.e. only changing latitude/longitude) as long as it keeps receiving hail from other drone. Once it stops receiving the hail from drone 1 it means there is another ship which is closer than drone 2. This means the attacker now has a circular projection of the area where the other user exists. By repeating this strategy with more drones he can narrow down the area and then sample it using oracle query. With high probability attacker should be able to find a user within budget."

Evaluation: Iowa State correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.4.2.8 TextCrunchr 2

A.2.4.2.8.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)
Iowa State provided a working exploit.

Evaluation: Iowa State correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.4.2.9 TextCrunchr 7

A.2.4.2.9.1 Question 033 (AC Time, Intended Vulnerable, Answered Yes)
Iowa State provided a working exploit.

Evaluation: Iowa State correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.5 Northeastern University

A.2.5.1 Northeastern Overview
Northeastern answered 10 questions with an 80% accuracy. Northeastern answered 9 SC time and AC time questions and only 1 SC space question. Northeastern had a 67% accuracy for SC time and an 83% accuracy for AC time questions.

Table A-32: Engagement 2 NEU Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	1	100	1	100
SC Time	3	2	67	2	67
SC Space/ Time	0	0	0	0	0
AC in Space	0	0	0	0	0
AC in Time	6	6	100	5	83
Total	**10**	**9**	**90**	**8**	**80**

Table A-33: Engagement 2 NEU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	8	7	**86**	6	**75**
Not Vulnerable	2	2	**100**	2	**100**
Yes Answer	7	7	**100**	6	**86**
No Answer	3	2	**67**	2	**67**

A.2.5.2 Northeastern Specific Responses

A.2.5.2.1 Blogger

A.2.5.2.1.1 Question 040 (AC Time, Intended Vulnerable, Answered Yes)

Northeastern responded that "there is a time vulnerability inside the URIVerifier.verify function. The application represents URIs as a package name for a Java class. It first replaces "/" with "." and then passes the URI to verify. The verify function breaks up the URI and ensures that the last URI element given in the string occurs at the end of the string.
The way the verify function parses the string is very inefficient. If you give it a URI with moderate depth, it slows down very quickly. For example, providing it the following URL causes the server to take 8 minutes 40 seconds on the reference hardware to verify.
GET /a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a/a"

Evaluation: Northeastern correctly identified the intended vulnerability in the challenge application and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.5.2.2 GabFeed 1

A.2.5.2.2.1 Question 007 (AC Time, Intended Vulnerable, Answered Yes)

Northeastern responded that "there is a time-based algorithmic complexity vulnerability. In the chat service, the program creates a linked list of messages submitted by users. New messages are appended to the end of the list, requiring an $O(n)$ traversal where n is the current size of the list. An attacker can append a large number of messages to this list with small message contents, thereby staying within the input budget while exceeding the resource usage limit."

Evaluation: The input budget is 400000 bytes. A standard HTTP POST command is composed of a POST, a requested page, an HTTP version, a host, and a content-length field, the minimum size for a POST request to add a new message to a GabFeed threat is 60 bytes. If a user was able to craft a POST command using only the bare minimum required components and reduced the size to 40 bytes, with a budget of 400,000 bytes the user would be able to post 10,000 messages. This is not sufficient to exceed the resource usage limit of 300 seconds. The intended vulnerability in the application was in the line breaking algorithm. The intended vulnerability required R&D Teams to recognize that users can set the width setting in the LineBreak.breakLines method. The method had O (n*w) asymptotic complexity where w is the

width setting and n is the number of words. By setting the width to be as least as long as the number of words the user can cause the algorithm to have O (n^2) behavior. It was later realized that the width setting was not necessary to cause the O (n^2) behavior and exceed the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.2.5.2.2.2 Question 010 (SC Time, Intended Vulnerable, Answered No)

Northeastern responded that "there is not a time-based side channel in the challenge. The static analyzer found that the authentication key only flows into the KeyExchange server which is the only location where modPow is called. The modPow method computes a value from the user's public key and the server's secret key. The method modPow introduces a *potential* side channel by only invoking the fastMultiply method in a loop when a bit in the private key is set. Timing modPow could in principle reveal the number of bits set in the private key, and this was confirmed using dynamic analysis. However," Northeastern were "unable to exploit this potential channel within the operational budget."

Evaluation: Northeastern correctly observed that there is a side channel in the modPow method which leaks the number of bits of value 1 in the server's private key. Northeastern did not correctly identify the intended vulnerability in the challenge application. The challenge application contained an SC Time vulnerability due to two sets of conditionals in the application. In the ModPow.modPow() method there is an if statement where the conditional is dependent on the value of each individual bit of the server's private key. Within the if statement there is a call to the OptimizedMultiplier.fastMultiply() method. This method contains an "if, else if, else" block where the conditional is dependent on the values of the variable "s", and the variable "base" which is the value submitted by an attacker. The side channel which leaks the server's private key is dependent on the differential processing times within the ModPow.modPow method and the OptimizedMultiplier.fastMultiply() method.

Post-Engagement Analysis Ruling: Incorrect

A.2.5.2.3 GabFeed 2

A.2.5.2.3.1 Question 032 (SC Time, Intended Not Vulnerable, Answered No)

Northeastern responded that "there is not a time-based side channel in the challenge program. Similarly to gabfeed_1, this program takes one of two similar branches in modPow depending on the value of each bit in the private key. However, the strength of this side channel appears to be less than the channel found in gabfeed_1." Northeastern concluded that they "do not believe that it can be exploited within the operational budget."

Evaluation: Northeastern correctly concluded that the challenge application did not contain an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.5.2.4 GabFeed 5

A.2.5.2.4.1 Question 009 (SC Time, Intended Not Vulnerable, Answered No)

Northeastern responded that "there is not a time-based side channel in the challenge program. Searching for a term causes a linear search of a flat text file. An adversary could search for all terms, record how long each of them takes, and then correlate this mapping against the observation of how long a victim's request takes to recover the victim's search term." Northeastern concluded that they "were not able to exploit the potential channel within the operational budget."

Evaluation: Northeastern correctly concluded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.5.2.5 Image Processor

A.2.5.2.5.1 Question 005 (AC Time, Intended Vulnerable, Answered Yes)

Northeastern responded that "there is a time-based algorithmic complexity vulnerability in the challenge program. The program contains two relevant methods for this vulnerability: Intensify.filter and Mathematics.intensify. The filter method takes an image to intensify, and invokes intensify. The intensify method takes the RGBA values of an area, and returns a new RGBA value. The return value is calculated using an exponentiation where the exponent is computed using RGB value components. By selecting input RGB values in the submitted image that maximize the exponent, the computation cost of the exponentiation operation increases such that the resource usage limit of the program can be exceeded while remaining under the input budget."

Evaluation: Northeastern correctly identified the intended AC Time vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.5.2.6 Law Enforcement Database

A.2.5.2.6.1 Question 019 (AC Time, Intended Not Vulnerable, Answered Yes)

Northeastern responded that "there is a time-based algorithmic complexity vulnerability in the challenge program. The server has a logging component that records all client activities to disk, which is an expensive task. An attacker can send a large number of search requests for the entire ID range (100000–40000000). Waiting for the server to process and log all requests as well as return responses exceeds the resource usage limit while remaining under the input budget."

Evaluation: The input budget of 1000 bytes allows for 111 search operations. Apogee initially ruled that application did not contain an intended AC Time vulnerability. Northeastern provided Apogee with an exploit which used the input budget to submit 111 search requests to the challenge application. These requests were sent faster than the challenge program could process them. In the exploit sent by Northeastern the script records the resource usage (time) from the moment the 111th search request packet is send to the moment the last byte of the server response

is received. Apogee observed that the time was approximately 76 seconds. This is greater than the resource usage of 30 seconds. The vulnerability discovered by Northeastern is an out of scope vulnerability because the branch of "slowness" is outside of the relevant java bytecode. In accordance with the Operational Definition, for discovering a vulnerability in the application which exceeded the resource usage limit, the post-engagement analysis ruling for this response was changed to correct.

Post-Engagement Analysis Ruling: Correct

A.2.5.2.7 SubSpace

A.2.5.2.7.1 Question 042 (SC Space, Intended Vulnerable, Answered Yes)

Response: Vulnerable

Northeastern responded that "there is a space-based side channel in the challenge program. Subspace stores user locations in a GeoMultiTrie, which as the name suggests is a trie where each node recursively partitions a region into quadrants until a minimum region size is reached (1e-4 on a side). An attacker can locate a real user by creating a new user and probing locations with that user using a search strategy that involves observations of the size of the database at /home/subspace/data/subspace.db. The search strategy begins by selecting a start point at (0*;* 0) and a starting region size that spans the entire space. At each step, the probe user's location is set to the center of the four quadrants of the current region size around the current center point. The closest probe will result in a *minimal* size increase in the database when the location is persisted. That probe becomes the new center point, and the region size is decreased by a half on each side. The reason that the minimal size is observed with the closest probe is because the closest probe should *share* some of the trie nodes with the target user. Other probes will not share nodes (or, in the general case with multiple users, will share *less* nodes with users that are further away). With a minimal region size of $1e-4$on a side, there are $\frac{180*360}{1e-4}$ possible positions. $\log_4 \frac{180*360}{1e-4} \approx 22$, meaning that the number of iterations is bounded by 22 steps."

Evaluation: Northeastern identified an SC Space vulnerability within the challenge application. The vulnerability discovered by Northeastern was the initial intended vulnerability within the challenge application. It was discovered in review that it is possible to triangulate a user's location by using the anonymous communication protocol of the challenge program. An attacker can create two users and move the users about the grid. If a message sent from one user arrives at the other then there are no other users closer to either user. An attacker can use this feature to triangulate the location of a user. The challenge question does not provide any passive operations; however, because the vulnerability discovered by Northeastern was the original intended vulnerability and demonstrates their research methods, Northeastern receives a post-engagement analysis ruling of correct for this response.

Post-Engagement Analysis Ruling: Correct

A.2.5.2.8 TextCrunchr 1

A.2.5.2.8.1 Question 021 (AC Time, Intended Vulnerable, Answered Yes)

Northeastern responded that "there is a time-based algorithmic complexity vulnerability for the challenge program. The loop that repeatedly unzips data from the input archive does not limit the

depth of the archives that it is unzipping. Therefore, if an attacker inputs a zip file that produces itself after zipping, the loop inside decompress can be made to run indefinitely."

Evaluation: Northeastern correctly identified the zip-bomb vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.5.2.9 TextCrunchr 7

A.2.5.2.9.1 Question 033 (AC Time, Intended Vulnerable, Answered Yes)

Northeastern responded that" there is a time-based complexity vulnerability in the challenge program. WordFrequencyProcessor counts the frequency of words in the input file using the class member method countWords. countWords uses a custom HashMap implementation, and [Northeastern] found that the get operation on the map was expensive. By supplying an input file containing repeated strings, an attacker could force the program to exceed the resource usage limit while remaining under the input budget. As an example, a proof-of-concept exploit caused the program to run for almost 18 minutes on the reference platform on a compressed input of size 57K before crashing."

Evaluation: Northeastern's response indicates that they identified the intended vulnerability in the challenge application. The application uses a hash table which is based on a red black tree. The tree is not balance on puts, therefore the challenge program is vulnerable to DOS. Provided an input size file with enough collisions the resource usage limit can be exceeded. The exploit description provided by Northeastern can be used to exceed the resource usage limit. Northeastern claimed that they created a "proof-of-concept" exploit. As detailed in the most recent Operational Definition document for future Engagements, a working exploit which exceeds the resource usage limit under the input budget is sufficient. Northeastern did not include this exploit file in their delivery however their descriptions of the vulnerability and exploit are accurate.

Post-Engagement Analysis Ruling: Correct

A.2.6 University of Maryland

A.2.6.1 UMD Overview

UMD answered 11 question with a 64% accuracy. UMD responded to more questions in E1 with a greater accuracy. UMD's E1 scores were bolstered by the fact that they identified an unintended vulnerability in all of the TextCrunchr variants. This unintended vulnerability was removed in Engagement 2 and UMD did not identify any of the intended TextCrunchr vulnerabilities. In Engagement 1 UMD stated that a lot of their responses were heavily aided by manual analysis. UMD answered 7 SC time questions. Many of the SC time questions UMD answered dealt with the same key exchange vulnerability which was present in some variants of GabFeed and SnapBuddy.

Table A-34: Engagement 2 UMD Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	0	0	0	0	0
SC Time	7	4	57	4	57
SC Space/ Time	1	1	100	1	100
AC in Space	0	0	0	0	0
AC in Time	3	3	100	2	67
Total	**11**	**8**	**73**	**7**	**64**

Table A-35: Engagement 2 UMD Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	7	6	**86**	5	**71**
Not Vulnerable	4	2	**50**	2	**50**
Yes Answer	8	6	**75**	5	**63**
No Answer	3	2	**67**	2	**67**

A.2.6.2 UMD Specific Responses

A.2.6.2.1 Blogger

A.2.6.2.1.1 Question 040 (AC Time, Intended Vulnerable, Answered Yes)

UMD responded that they "ran static information flow analysis (that tracks both explicit and implicit flows) using the main method in fi.iki.elonen.JavaWebServer as the entry point and specifying the first parameter of method fi.iki.elonen.JavaWebServerPlugin.canServeUri as a tainted source. The static information flow analysis reported 26 tainted loop variables from 20 methods. Among them, the method fi.iki.elonen.URIVerifier.verify has three loops that were reported as tainted and uses complicated nested loop structures." UMD then used "the fuzzing tool to fuzz the method given the input format (i.e., URI String). The fuzzing tool exposed the algorithmic complexity vulnerability in time, resulting in the program running longer than 10 minutes (well exceeds the 300 seconds threshold)." UMD's "dynamic tool observed that the method fi.iki.elonen.URIVerifier.verify took up the majority of the running time."

Evaluation: UMD correctly identified the intended AC Time vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.6.2.2 GabFeed 1

A.2.6.2.2.1 Question 010 (SC Time, Intended Vulnerable, Answered No)

UMD responded that they "ran static information flow analysis using all the main methods in the program. The methods containing loops that were affected by both tainted and secret sources include ModPow.modPow and OptimizedMultiplier.standardMultiply." UMD "ran the static path-based bound analysis on the modpow functionality and its subsequent calls. The analysis

generates paths for modpow method that expose different bounds. This method, however, uses randomization to obfuscate the running time differences. The invocation of Random.nextDouble appears on the bound, indicating the influence of randomization."

Evaluation: UMD did not identify the intended vulnerability in the application. UMB correctly observed that the ModPoe.modPow and OptimizedMultiplier.standardMultiply methods contained loops that were affected by both tainted and secret sources. In the ModPow.modPow() method there is an if statement where the conditional is dependent on the value of each individual bit of the server's private key. Within the if statement there is a call to the OptimizedMultiplier.fastMultiply() method. This method contains an "if, else if, else" block where the conditional is dependent on the values of the variable "s", and the variable "base" which is the value submitted by an attacker. The side channel which leaks the server's private key is dependent on the differential processing times within the ModPow.modPow method and the OptimizedMultiplier.fastMultiply() method. In GabFeed 1, the implementation of the for-loop using the does not remove the side channel.

Post-Engagement Analysis Ruling: Incorrect

A.2.6.2.2.2 Question 029 (SC Time, Intended Not Vulnerable, Answered No)

UMD responded that they "ran static path-based bound analysis on the method which both password typed by the user (tainted) and the secret password reach (i.e., com.cyberpointllc.stac.webserver.User.passwordsEqual). The passwordsEqual method compares between the tainted and secret passwords. For all the paths that the analysis generated that executes the statements inside the loop body, the bounds are indistinguishable." UMD "therefore believes the secret password cannot be leaked in time so that an attacker cannot impersonate a user via the username/password."

Evaluation: UMD correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.6.2.3 GabFeed 2

A.2.6.2.3.1 Question 032 (SC Time, Intended Not Vulnerable, Answered Yes)

UMD responded that they "ran static information flow analysis using all the main methods in the program. The methods containing loops that were affected by both tainted and secret sources include ModPow.modPow and OptimizedMultiplier.standardMultiply." They "ran the static path-based bound analysis on the modpow functionality and its subsequent calls. The method com.cyberpointllc.stac.math.OptimizedMultiplier.standardMultiply exposes different bounds for two paths based on the result of invoking method BigInterger.testBit on a tainted BigInteger." UMD concluded that "the running time of the modpow method is dependent on secret BigInteger's results on invoking testBit."

Evaluation: The application did not contain an intended SC Time vulnerability; however, it contained a weak SC Time vulnerability which leaks the hamming weight (number of 1's) of the server's private key.

Post-Engagement Analysis Ruling: Incorrect

A.2.6.2.4 GabFeed 3

A.2.6.2.4.1 Question 036 (AC Time, Intended Vulnerable, Answered Yes)

UMD responded that they "ran static bound analysis on the method PageUtils.formatLongString, and found that the number of split tokens in the input data impacts the complexity of an already highly complex line breaking code."

Evaluation: The line breaking algorithm in this GabFeed variant was not vulnerable; the sorting algorithm was. The algorithm typically has O (n log n) behavior but can have O (n^2) behavior if the number of items being sorted is divisible by 8.

Post-Engagement Analysis Ruling: Incorrect

A.2.6.2.4.2 Question 039 (SC Time and Space, Intended Vulnerable, Answered Yes)

UMD responded that "through manual analysis they identified HTTP/S handlers in the source which were related to the question description (com/cyberpointllc/stac/gabfeed/handler/SearchHandler.java, in particular)." UMD "identified a branch in handlePost (where user search term queries are handled) on the guard 'if (this.dailyTerms.contains(searchTerm)). If there is a difference in space and time on these two branches, then an attacker who can observe queries can identify that a user has searched for a daily ('special') term. Using" their "bound analysis," UMD "identified that 'com.cyberpointllc.stac.linebreak.LineBreak.breakLines(Ljava/lang/String;)Ljava/util/List;' has complex and large bounds, corresponding to its control flow." UMD "also used call graph generation to verify that this method is reachable from the then-branch mentioned above. In addition, there is extra markup appended to the response, stored in "specialContents" in the then-branch. Using the attacker's 0-cost active operations," UMD "believes that they can identify a correlation between space or time and daily search terms (assuming that SearchHandler.getInfoText uniquely identifies each search term). In this way," they "can also identify which search term the user is searching for, using pre-cached results based on attacker's observations of daily search terms." UMD "used an off the shelf decompiler (JAD and CFR) to decompile the challenge program jar for manual analysis." They identified "interesting parts of the program manually, and ran call graph generation (adapted from WALA) and bound analysis as described above."

Evaluation: UMD correctly observed that the "if (this.dailyTerms.contains(searchTerm))" statement is used to perform additional operation in the case of special search term. In the case of a special search term search, the challenge program reads from a file containing the special search terms and adds additional information to the response. Using the provided database this behavior of the application can be used to distinguish special search term searches from all other GabFeed interactions. UMD correctly noted an attacker can use their 0-cost active operations to force a correlation between the size of special search term responses and other GabFeed responses. It is true that an attacker can user their active operations to increase the response packet size for the special terms; however, the provided database should be sufficient to allow an attacker to map the size of special search terms to a specific special search term once the user interaction is classified as a special term search. UMD identified the intended vulnerability and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.6.2.5 Image Processor

A.2.6.2.5.1 Question 005 (AC Time, Intended Vulnerable, Answered Yes)

UMD “ran static information flow analysis (that tracks both explicit and implicit flows) using the main method in com.stac.Main as the entry point and specifying the first parameter of method com.stac.Main.main as a tainted source.” UMD “dealt with object instantiations using reflective Class.newInstance, the analysis reported 45 tainted loop variables from 17 methods. Among the methods reported by taint analysis, three color detector algorithms (i.e., com.stac.image.algorithms.detectors.RedDetector.detect, com.stac.image.algorithms.detectors.BlackDetector.detect, and com.stac.image.algorithms.detectors.WhiteDetector.detect) involve nested loops.” UMD “ran static bound analysis on these methods. The analysis obtained the algorithmic complexity of each method is bounded by width * height, where width and height of the BufferedImage are the result of calling java.awt.image.BufferedImage.getWidth() and java.awt.image.BufferedImage.getHeight(), respectively. Given the fact that width and height of an image within the input budget can be large, the image feature characterization invoking these methods can be slow.” UMD confirmed “this report by generating a PNG file and running the program with classification command well over 1080 seconds.”

Evaluation: The challenge application restricts input images to 250,000 pixels (com.stac.image.ImageProcessing class). The input provided by UMD violates that restriction. The application uses the java.awt.image.BufferedImage class to process all input images. This class takes longer to process images with more pixels. This class is called for all input images. The resulting BufferedImage object is used to check the application’s input guard. If the application recognizes that the input image exceeds 250,000 pixels an exception is thrown. According to the Operational Definition document for STAC, the branch for ‘slowness’ in complexity vulnerabilities must occur inside the challenge program. In this case the branch for ‘slowness’ occurs in the java.awt.image.BufferedImage class which is an external library. The vulnerability identified by UMD is not valid by this definition. The intended vulnerability for the challenge program is within the Mathematics.intensify function. The filters.Intensify.filter function iterates through the pixels of a provided image and calls the Mathematics.intensify function for each pixel. The Mathematics.intensify function contains a loop in its *exp* function. The termination point of this loop depends on the value of the pixel provided to the Mathematics.intensify function. An input image of size 500 pixels by 500 pixels containing the rgb value which maximizes the number of loops performed in the *exp* is sufficient to exceed the resource usage limit. UMD identified an out of scope vulnerability and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.6.2.6 Law Enforcement Database

A.2.6.2.6.1 Question 026 (SC Time, Intended Vulnerable, Answered Yes)

UMD “decompiled the jar and manually inspected the source to determine that channelRead0 involves both tainted and secret input.” UMD “had to determine this through manual inspection because the taint analysis relies on complete call graph, whose size was inflated by the netty-all-4.0.29.Final.jar 3rd party library. The callgraph then became too large for the taint analysis to scale reasonably (> 48 hours of runtime on standard apple hardware). The server.UDPServerHandler.channelRead0 that takes the tainted DatagramPacket and queries the

secret database is vulnerable to side channel attack in time from one of its cases (i.e., case 8 that deals with the search by a range of ids).” UMD “ran static path-based bound analysis on a modified code piece to confirm this finding and exploited an attack. Among multiple paths generated by the analysis, the bound of two paths (i.e., one that the tainted keys within the range always match the secret values and another that the tainted keys never match the secret values) is indistinguishable (i.e., range.size). Note that method sends a DataPacket if a tainted key is not secret; the same loop bound for the two paths indicates the frequency/time interval of receiving the packets can be significantly different depending on the secrecy of the key.” UMD “therefore designed an exploit by recording the time interval of receiving packet from a search range request and narrowing the range by inserting keys and observing the longer intervals between packets.”

Evaluation: UMD correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.6.2.7 SnapBuddy 2

A.2.6.2.7.1 Question 015 (SC Time, Intended Vulnerable, Answered Yes)

UMD “ran static information flow analysis using all the main methods in the program. The methods containting loops that are affected by both tainted and secret sources include ModPow.modPow and OptimizedMultiplier.standardMultiply.” UMD “ran static path-based bound analysis on com.cyberpointllc.stac.math.ModPow.modPow, which analyzes the bound of three distinct paths of the method. For the two paths that always iterate over the loop via different branches inside the loop (i.e., the testBit invocation always return true/false on the secret BigInteger, their bounds differ by width * B(com.cyberpointllc.stac.mathOptimizedMultiplier.fastMultiply) * B(java.math.BigInteger.mod), where width is the result of the secret BigInterger invoking bitLength and B(M) denotes the bound of the method M. The significant difference in terms of the bounds on two paths within the modpow loop indiates the high probability of leaking secret information (as the branch is guided by secret BigInterger's result on testBit) in time.”

Evaluation: The challenge application contained an SC Time vulnerability due to two sets of conditionals in the application. In the ModPow.modPow() method there is an if statement where the conditional is dependent on the value of each individual bit of the server’s private key. Within the if statement there is a call to the OptimizedMultiplier.fastMultiply() method. This method contains an “if, else if, else” block where the conditional is dependent on the values of the variable “s”, and the variable “base” which is the value submitted by an attacker. The side channel which leaks the server’s private key is dependent on the differential processing times within the ModPow.modPow method and the OptimizedMultiplier.fastMultiply() method. UMD identified the vulnerability in the modPow method but did not identify the vulnerability in the OptimizedMulitplier.fastMultiply method. UMD’s response was initially given a post-engagment analysis ruling of incorrect. We (Apogee) did not believe that the vulnerability identified by UMD was sufficient because the vulnerability in the OptimizedMultiplier.fastMultiply method allows an attacker to craft an input which coupled with the side channel in the modPow method leaks the server’s private key. After the initial Engagement 2 report was released, UMD provided additional information which showed that their tools identified the OptimizedMultiply.fastMultiply and the ModPow.modPow methods as potentially vulnerable. UMD then consulted with a Cybersecurity expert who stated that the method was indeed

vulnerable. The additional information provided by UMD showed that they had sufficiently identified the intended side channel vulnerability within the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.6.2.8 SnapBuddy 3

A.2.6.2.8.1 Question 018 (SC Time, Intended Not Vulnerable, Answered Yes)

UMD "identified a method that checks the session id from a cookie. This method uses a hashmap that ultimately calls a TreeNode.find method." UMD's "static bound analysis shows that each trail through the method is constant time, but that some make a recursive call to find and some do not. Because of this output, it is possible that an attacker may be able to observe timings differences depending on whether the session id is in the hash map."

Evaluation: UMD reported an unintended SC Time vulnerability in the TreeNode.find() method. The WebSessionService contains the getSession method which uses the sessionID string determined from a user's session cookie to determine if a user's session exists. If an attacker can determine a user's sessionID, the attacker can use the information to impersonate a user. The challenge program uses a com.cyberpointllc.stac.hashmap.HashMap to keep track of user sessions by sessionID. The getSession method contains a conditional "if (sessionId != null && sessions.containsKey(sessionID)." This conditional calls the HashMap.containsKey method which calls the HashMap.get() method with a user's sessionID as an argument. The HashMap class maintains sessionIDs in buckets with collisions handled through linked lists. Once such linked list reaches TREEIFY_THRESHOLD (8 items), the linked list is replaced with an unbalanced tree. When the HashMap.get() is called if the hashed entry is maintained as a linked list, the method iterates through the linked list until it finds the sessionID or reaches the end of the linked list. If the hashed entry is maintained as an unbalanced tree, the method iterates through the unbalanced tree to find the provided sessionID. In the case where the user's sessionID is not hashed to the same bin as another sessionID, there does not appear to be a side channel which leaks the sessionID of the user. In the case where a user's sessionID is hashed to the same bin whether the bin is maintained as a linked list or an unbalanced tree, there is a side channel which could potentially leak the position of the sessionID in either the linked list or the unbalanced tree. The EL does not believe that this side channel is strong enough to leak the sessionID of a user within the operational budget.

Post-Engagement Analysis Ruling: Incorrect.

A.2.6.2.9 SubSpace

A.2.6.2.9.1 Question 041 (SC Time, Intended Not Vulnerable, Answered No)

UMD "ran the static path-based bound analysis on the logic of comparing tainted and secret password strings, specifically com.example.util.Crypto.isEqual([BII[BII)Z. For all the feasible paths inside the loop, the bounds are not different." UMD therefore concluded that "the password could not be leaked via timing channel and a third party cannot impersonate a user."

Evaluation: UMD correctly concluded that the challenge application did not contain an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.7 University of Utah

A.2.7.1 Utah Overview

Utah answered 10 questions with an 80% accuracy. Utah focused entirely on complexity questions answering 2 AC space and 8 AC time questions. Utah had a 100% accuracy for AC space vulnerabilities.

Table A-36: Engagement 2 Utah Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	0	0	0	0	0
SC Time	0	0	0	0	0
SC Space/ Time	0	0	0	0	0
AC in Space	2	2	100	2	100
AC in Time	8	6	75	6	75
Total	**10**	**8**	**80**	**8**	**80**

Table A-37: Engagement 2 Utah Vulnerablity Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	7	5	**71**	5	**71**
Not Vulnerable	3	3	**100**	3	**100**
Yes Answer	5	5	**100**	5	**100**
No Answer	5	3	**60**	3	**60**

A.2.7.2 Utah Specific Responses

A.2.7.2.1 GabFeed 3

A.2.7.2.1.1 Question 036 (AC Time, Intended Vulnerable, Answered Yes)

Utah responded that “if a room contains 104 threads (or larger multiples of 8), then retrieving a list of the threads in that room (using the URL https://localhost:8080/room/N for room number N) exceeds the resource usage limit of 300 seconds.” Utah added that they “can ensure a room contains 104 threads by repeatedly running this command `curl -s -F threadName="a" -F messageContents="b" -b cookies.txt --insecure https://localhost:8080/newthread/N`, where N is the room number. Each of these `curl` commands sends a packet of size 470 bytes. If there is no room with fewer than 104 threads, just keep adding threads until the number of threads in the room is the next higher multiple of 8. Therefore the maximum number of `curl` requests that must be sent is 104, resulting in the sum of PDU sizes of 104 * 470 = 48,880 bytes of HTTP requests, which is within the budget of 400,000 bytes.”

Evaluation: The details provided by Utah are sufficient to create a working exploit for this challenge program. The intended vulnerability in the application is in the sort function. When the

list of items to be sorted is divisible by 8, the application exceeds the resource usage limit given at least 40 items.

Post-Engagement Analysis Ruling: Correct

A.2.7.2.2 Image Processor

A.2.7.2.2.1 Question 005 (AC Time, Intended Vulnerable, Answered Yes)

Utah responded that “using the default tags on the training images, classifying the file `image_processor_question_30_attack.jpg` (a JPEG file of dimensions 528 x 453 pixels, for which the RGB values of each pixel are 0, 30, 0 (summing to 30)) results in a running time of over 1080 seconds measured from the submission of the last user command to the completion of the application's processing of the command.”

Evaluation: Utah provided a working exploit and the description provided by Utah shows they identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.7.2.2.2 Question 030 (AC Space, Intended Not Vulnerable, Answered No)

Utah “found no evidence of an algorithmic complexity vulnerability in space that would cause the memory usage of the challenge program to exceed the resource usage limit given the input budget.”

Evaluation: The application did not contain an intended AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.7.2.3 TextCrunchr 1

A.2.7.2.3.1 Question 021 (AC Time, Intended Vulnerable, Answered No)

Utah “found no evidence of an algorithmic complexity vulnerability in time that would cause the challenge program's real runtime (on the reference platform) to exceed the resource usage limit given the input budget.”

Evaluation: Utah did not identify the intended AC Tim vulnerability in the application. The challenge application is vulnerable to a zip bomb. The ZipDecompressor will continuously unzip data from an archive without a limit on the depth of the archive. Given an input zip file that reproduces itself after zipping the resource usage limit can be exceeded.

Post-Engagement Analysis Ruling: Incorrect

A.2.7.2.4 TextCrunchr 2

A.2.7.2.4.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Utah responded that “the HashMap implementation in TextCrunchr 2 performs poorly when a large number of hashcode collisions occur. Any input that contains a large number of words that hash to the same hashcode will trigger quadratic behavior.”

Evaluation: Utah provided a working exploit and a description of the intended vulnerabilities in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.2.7.2.5 TextCrunchr 3

A.2.7.2.5.1 Question 011 (AC Time, Intended Vulnerable, Answered Yes)

Utah responded that “any input with a large number of unique words will run slowly, provided the ‘WordFreq’ analysis is run. This is because the sorting algorithm runs a quicksort after running a mergesort, and sorting an already sorted list leads to quadratic behavior in quicksort.”

Evaluation: Utah provided a working exploit and a description of the intended vulnerabilities in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.2.7.2.6 TextCrunchr 5

A.2.7.2.6.1 Question 017 (AC Space, Intended Not Vulnerable, Answered No)

Utah “found no evidence of an algorithmic complexity in space that would cause the challenge program's memory usage to exceed the resource usage limit given the input budget.”

Evaluation: Utah correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.7.2.6.2 Question 024 (AC Time, Intended Not Vulnerable, Answered No)

Utah “found no evidence of an algorithmic complexity vulnerability in time that would cause the challenge program's real runtime (on the reference platform) to exceed the resource usage limit given the input budget.”

Evaluation: Utah correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.7.2.7 TextCrunchr 6

A.2.7.2.7.1 Question 008 (AC Time, Intended Vulnerable, Answered Yes)

Utah responded “that the sorting algorithm performs poorly for certain inputs with a length that is a multiple of 3. Inputs that contain a number of unique words that is a multiple of 3 trigger poor performance of the sorting algorithm.”

Evaluation: Utah correctly identified the intended vulnerability in the challenge application and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.7.2.8 TextCrunchr 7

A.2.7.2.8.1 Question 033 (AC Time, Intended Vulnerable, Answered No)

Utah “found no evidence of an algorithmic complexity vulnerability in time that would cause the challenge program's real runtime (on the reference platform) to exceed the resource usage limit given the input budget.”

Evaluation: Utah did not identify that TextCrunchr 7 uses a hash table which is based on a red black tree. The tree is not balance on puts, therefore the challenge program is vulnerable to DOS. Provided an input size file with enough collisions the resource usage limit can be exceeded.

Post-Engagement Analysis Ruling: Incorrect

A.2.8 Vanderbilt University

A.2.8.1 Vanderbilt Overview

Vanderbilt responded to all 42 questions with an 83% accuracy. Vanderbilt correctly answered all SC space questions and missed 1 AC time question. Vanderbilt incorrectly answered 4 SC Time and 2 AC space questions. Vanderbilt answered as many questions as the control team with greater accuracy primarily in the SC Space and AC Time categories. In the Graph Analyzer challenge program Vanderbilt failed to identify either the AC Time or the AC Space vulnerabilities. Vanderbilt outperformed all Blue Teams by answering more questions and had the greatest accuracy.

Table A-38: Engagement 2 Vanderbilt Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	7	7	100	7	100
SC Time	10	7	70	6	60
SC Space/ Time	1	1	100	1	100
AC in Space	8	6	75	6	75
AC in Time	16	15	94	15	94
Total	**42**	**36**	**86**	**35**	**83**

Table A-39: Engagement 2 Vanderbilt Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	23	19	**83**	18	**78**
Not Vulnerable	19	17	**89**	17	**89**
Yes Answer	21	19	**90**	18	**86**
No Answer	21	17	**81**	17	**81**

For Subspace question 42, Vanderbilt correctly identified the intended vulnerability but responded “no” because they did not believe that it fit the required STAC vulnerability. This challenge question was adjusted from the initial vulnerability. Vanderbilt correctly identified the intended vulnerability and verified that it was sufficiently strong. Vanderbilt only ruled out the vulnerability on a definitional basis therefore the response was given a post-engagement analysis ruling of correct.

A.2.8.2 Vanderbilt Specific Responses

A.2.8.2.1 Blogger

A.2.8.2.1.1 Question 016 (AC Space, Intended Not Vulnerable, Answered No)

Vanderbilt "applied their Janalyzer tool to look for vulnerabilities. The tool showed an inter-procedural 3-level nested loop. Vanderbilt observed that the exit condition of the loop is modified inside the loop." The method identified was the fi.iki.elonene.URIVerifier.verify() method. Vanderbilt proceeded to analyze the method for AC vulnerabilities. Vanderbilt performed manual analysis and verified that "the loop exit condition (whether a LinkedList is empty) is modified inside the loop." Vanderbilt tested the identified method for an AC Space vulnerability and concluded that the method did not contain such a vulnerability. Vanderbilt performed other analysis to test the Space utilization of the application and concluded that the application was not vulnerable.

Evaluation: Vanderbilt did not identify an AC Space vulnerability within the application. There was not an intended AC Space vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.1.2 Question 040 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt analyzed the application using their Janalyzer tool and identified the fi.iki.elonene.URIVerifier.verify() method and potentially vulnerable to an AC Time vulnerability. Vanderbilt observed that the method contains a "conditional on the length of the URI that pushes an element to the stack." Vanderbilt noted that the "runtime appears to be exponential with respect to the length of the URI." Vanderbilt observed that "a long enough string" of "lowercase characters" causes an "exponential" "behavior". Vanderbilt provided a working exploit.

Evaluation: Vanderbilt correctly identified the intended AC Time vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.2 GabFeed 1

A.2.8.2.2.1 Question 007 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt used their Visualizer, VisualVM, Clprof, and SPF WorstCaseAnalyzer tools to identify the vulnerability within the application. Vanderbilt looked at the source code and used their Visualizer to identify "a suspicious loop in com.cyberpointllc.stac.linebreak.LineBreak()" method that "is quadratic in the size of the variable 'count'." Vanderbilt used the SPF WorstCaseAnalyzer tool to write a driver that generated "an array of symbolic strings".

Vanderbilt confirmed that the "line break algorithm is quadratic in the size of the input array of words." Vanderbilt observed that only the length of the array contributes to LineBreak's performance. Vanderbilt provided a graph of the worst case prediction model.

Vanderbilt observed that "in the line break algorithm, the 'count' variable is the number of words in the input string. The input string can be various things depending on the user interaction since

LineBreak is used in multiple cases. In the simplest case, the input string is the string of a new message submitted by a user." Vanderbilt provided an input string composed of "a a a a" repeated.

Evaluation: Vanderbilt correctly identified the AC Time vulnerability within the challenge application

Post-Engagement Analysis Ruling: Correct

A.2.8.2.2.2 Question 010 (SC Time, Intended Vulnerable, Answered No)

Vanderbilt identified an SC Time vulnerability within the challenge application in the ModPow class. Vanderbilt noted that the "vulnerability reveals to the attacker the information about the private key used in the server's authentication." Vanderbilt observed that "there is significant noise that obscures the channel and by inspecting the code they observed that there is programmatic randomness added to the mod pow multiplication functions.

A channel that reveals the number of 1's in the secret is not strong enough to reveal the secret, even without noise. For example, if the secret has 128 bits, in the worst case 64 bits of the secret are 1 which means that there are (128!)/((64!)(64!)) possible secrets impossible to reveal within the budget."

Vanderbilt attempted a known attack to reveal one of the prime factors. Vanderbilt observed that the attack did not work and concluded that the challenge application did not contain an SC Time vulnerability.

Evaluation: Vanderbilt identified the vulnerable mod pow method within the challenge application. Vanderbilt concluded that the artificial noise obscures the side channel. This is not correct. The artificial noise obscures but does not remove the side channel. Vanderbilt was incorrect in their analysis of the strength of the side channel.

Post-Engagement Analysis Ruling: Incorrect

A.2.8.2.2.3 Question 029 (SC Time, Intended Not Vulnerable, Answered No)

Vanderbilt responded that "a straightforward way to achieve" the "goal of impersonating a given user is to discover their password and simply log in with their information." Vanderbilt "performed a symbolic execution of the password comparison function using SPF and discovered that there is a single instruction difference in timing measurements which reveals the number of matching positions in the hashed password and the hashed user input password. Thus there is a side channel which could reveal the hashed password."

Vanderbilt stated that the identified side channel relies on being able to detect timing differences for a single instruction and that the side channel only reveals information about the hashed password. Vanderbilt provided a details of potential exploit for this side channel and stated that the exploit requires the secret key used in the DES encryption. Vanderbilt discussed several methods of identifying the DES encryption key. Vanderbilt identified a potential SC Time vulnerability in the DES encryption where the application performs "a bit shift followed by an 'and' operation:

```
this.dst = (this.dst << 1 | (this.src >> srcPos &0x1L"
```

Vanderbilt analyzed this second side channel and concluded that it was not sufficiently strong.

Vanderbilt discussed another potential SC. Vanderbilt stated that this SC Time uses cache memory to determine the secret. Vanderbilt concluded that this attack would violate the operational budget. Vanderbilt concluded that the challenge application did not contain an SC Time vulnerability.

Evaluation: Vanderbilt correctly observed that there was not an intended SC Time vulnerability within the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.2.4 Question 038 (AC Space, Intended Not Vulnerable, Answered No)

Vanderbilt ran their "SPF WorstCaseAnalyzer on the sorting algorithm in the application." The tool showed that "memory consumption is approximately linear with respect to the size of the list." Vanderbilt concluded that the sorting algorithm was likely not vulnerable. Vanderbilt ran their "Fuzzer tool on the sorting algorithm using randomly generated strings and found no evidence of a space complexity vulnerability." Next Vanderbilt analyzed the LineBreak class within the application. Vanderbilt observed that the class "splits up paragraphs first, then for each paragraph, the breakLine method is called." Vanderbilt noted that "the worst case behavior could be triggered by sending as many as possible paragraphs of 5 words each." Vanderbilt observed that while this consumed time it did not "consume an extraordinary amount of memory". Vanderbilt concluded that the application did not contain an AC Space vulnerability

Evaluation: Vanderbilt correctly concluded that the application did not contain an intended AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.3 GabFeed 2

A.2.8.2.3.1 Question 006 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt used their Janalyzer, VisualVM, and SPF WorstCaseAnalyzer tools. Vanderbilt stated that "there is a time complexity vulnerability in the hashmap implementation (com.cyberpointllc.stac.hashmap.HashMap)." Vanderbilt used their SPF WorstCaseAnalyzer to confirm
that it is possible to construct inputs which cause look-ups to be performed in linear time." Vanderbilt stated that "the hashmap implementation is used in various placed in GabFeed – notably for indexing all words of all messages among others in the GabFeed database model. Specifically, the instance variables indices in com.cyberpointllc.stac.gabfeed.persist.GabDatabase uses an instance of the vulnerable hashmap implementation. […] For indices, the hashmap implementation maps strings to GabIndexEntry instances. The keys of a hashmap are computed based on a hashcode(). […] Hash collisions are initially organized as a linked list structure, but if the bin count exceeds 8 entries, the hashmap implementation "treeifies" the list into a binary tree. The vulnerability lies in the binary tree implementation which does not perform rebalancing. In the worst case the "treeification" can yield a linked list when items for example are inserted in sorted order (or the opposite of sorted order). Because" the user "can control the strings added to the hashmap it is very easy to

generate the worst case configuration of the hashmap where lookups are O(n)". Vanderbilt provided a working exploit which exceeded the resource usage limit.

Evaluation: Vanderbilt correctly identified the intended vulnerability within the application. Vanderbilt provided a working exploit of the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.3.2 Question 032 (SC Time, Intended Not Vulnerable, Answered No)

Vanderbilt "found a timing channel vulnerability in the method modpow of the class ModPow. This vulnerability reveals information about the private key used in the server's authentication. By manual inspection of the modular exponentiation function," Vanderbilt observed that "there is a branch that is only taken when the i^{th} bit of the secret key is 1." Vanderbilt stated "there is a side channel which reveals the number of 1 bits in the secret." Vanderbilt "performed a sample-based profiling of the modular exponentiation functions found in all versions of the GabFeed problems in which they varied the number of 1's in the secret and recorded the computation time." The results of this profiling showed that "there is little noise that obscures the channel, but that execution time seems constant even when varying the exponent." Vanderbilt concluded that the side channel was not sufficiently strong.

Evaluation: Vanderbilt correctly concluded the challenge application did not include an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.4 GabFeed 3

A.2.8.2.4.1 Question 023 (SC Time, Intended Vulnerable, Answered Yes)

Vanderbilt responded that "in order to determine how many special search terms a user searched for, it would be sufficient to find an SC Time vulnerability which allows an attacker to identify the search for a single special search term." Vanderbilt "manually inspected the code to determine that the time taken per search request is proportional to both whether or not the term is a special search term and how many hits the term has in the database." Vanderbilt experimented with searches for special search terms and non-special search terms. Vanderbilt observed that the mean separation between the two groups was 18 ms. Vanderbilt stated that while the mean separation was sufficient to distinguish the two categories the noise in the side channel was too high to allow for a 90% success probability in the worst case.
Vanderbilt noted that an AC Time vulnerability in the sort method occurs when the length of the list being sorted is divisible by 8. Vanderbilt used the 0 cost of active operations provided in the available operations to pad the database with special search terms.

Evaluation: Vanderbilt correctly identified the intended vulnerability within the application. Vanderbilt did not explicitly state that special search terms take longer because the server reads additional information form a text file prior to responding to a special term search. Vanderbilt did observe the separation in the mean response time of special search terms from non-special search terms. It was intended that the separation in mean response time between special search term searches and other GabFeed operations would be sufficient. It was also intended that the

provided data base on the reference platform had small enough variability for a sufficient side channel.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.4.2 Question 036 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt used their decompiler and Visualizer tools to observe that the application uses "a number of HTTP handlers." Vanderbilt noted that the challenge question "restricts the interaction to a single user" and concluded that "the vulnerability could pertain to how search, chat rooms, threads, and messages are managed. Chat rooms have threads, which […] have messages;" Vanderbilt stated that "experimenting with how messages are handled seemed to be the easiest approach." Vanderbilt created a script to profile "the HTTP response time" when a number of messages are sent. Vanderbilt observed that "certain number of messages […] (17, 25, 33)" cause the response time to be high.

Vanderbilt used this observation in conjunction with their VisualVM tool to understand the time-consuming operations within the application. Vanderbilt observed from the collected data that the sorting algorithm in the com.cyberpointllc.stac.sort.Sorter class was a "major contributor to the high response times." Vanderbilt noted that this class "sorts the messages present in the HTTP response."

Vanderbilt used their SPF WorstCaseAnalyzer tool to observe that the sorting algorithm is slow "for an input list size of 16." This observation led Vanderbilt to identify a vulnerable part of the application: the sorting mechanism has an anomalous behavior if the list is divisible by 8.

Vanderbilt analyzed the MapDV database that stores message to understand how to use the vulnerability. They observed that the database contained "7 messages for the default user, thus for every (n+7) % 8 == 0 number of messages sent, the server responds unusually slow[ly]."

Evaluation: Vanderbilt correctly identified the AC Time vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.4.3 Question 039 (SC Time and Space, Intended Vulnerable, Answered Yes)

Vanderbilt responded that in response to Question 23 they had demonstrated an SC Time vulnerability which allows an adversary to determine whether a user operation is a special search term test. Vanderbilt was left with differentiating between different special search term searches. Vanderbilt "ran the dependency slicer in Janalyzer on the packet returned by the handlePost() method of the com.cyberpointllc.stac.gabfeed.SearchHandler class to determine that the size of the packet is proportional to the number of bytes in both the size of the text associated with the special term and the side of the hits on the public threads." Vanderbilt tested this, collected data, and created a mapping between special search terms and the response packet size. Vanderbilt observed that "each size corresponds to exactly one special search term." Vanderbilt added that in the case where a user modifies the public database the attacker can observe the public database for changes and adjust the mapping to account for changes made by the user.

Evaluation: Vanderbilt correctly identified the intended SC Time and Space vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.5 GabFeed 4

A.2.8.2.5.1 Question 003 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt responded that the application contained "the same complexity vulnerability as GabFeed 2. The only difference between GabFeed 4 and GabFeed 2 is that GabFeed 4 converts the collision list of the hash table into a balanced binary tree while GabFeed 2 makes a conversion to a potentially unbalanced binary tree." Vanderbilt observed that "subsequent addition of elements to the tree does not trigger rebalancing as it does for GabFeed 1, 3, and 5. "Treeifying" the list of collisions happens when the linked list contains more than 8 elements." Vanderbilt noted that "this initial "treeification" does not really help on performance and is susceptible to the same attack."

Vanderbilt stated that "there is a time complexity vulnerability in the hashmap implementation" of the challenge application in the com.cyberpointllc.stac.hashmap.HashMap class.

Vanderbilt used their SPF WorstCaseAnalyzer to show "that it is possible to construct inputs such that look-ups are always performed in linear time."

Vanderbilt stated that "for indices, the hashmap implementation maps Strings (e.g. words in submitted GabFeed messages) to GabIndexEntry instances. The keys of a hashmap are computed based on hashcode()." Vanderbilt noted that "String.hashcode() is not secure because it is easy to generate different strings with the same hash." Vanderbilt observed that hash "collisions are initially organized as a linked list structure but if the bin count exceeds 8 entries the hashmap implementation 'treeifies' the list into a binary tree." Vanderbilt stated that "the vulnerability lies in the binary tree implementation which does not perform rebalancing." Because the adversary "can control the strings added to the hashmap, which can simple be the words in a message submitted to GabFeed, it is easy to generate the worst case configuration of the hashmap where look-ups are O (n)."

Vanderbilt states that the challenge application implementation results in a complexity of "O(n*m*w) where n is the number of words that hash to the same bin, m is the number of unique words in the message, and w is the size of the words." Vanderbilt created and provided a working attack.

Evaluation: Vanderbilt correctly identified the intended vulnerability within the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.6 GabFeed 5

A.2.8.2.6.1 Question 001 (SC Space, Intended Not Vulnerable, Answered No)

Vanderbilt identified a vulnerability which can leak the username of a user involved in a private chat. "When chats are started they are added to the database according to their id which is the number of chats at the time. The results in a monotonically increasing list of ids (0,1,2,…,n) where n is the total number of chats." An "attacker can start a session and send a GET request for chat 0, 1,2,… until a 404 page is reached for an invalid chat." Vanderbilt stated that "this means the attacker can observed the chat and display the names of the users involved in any chat they can access." Vanderbilt observed that the handleGet() method of the ChatHandler class does not check whether or not the user requesting the chat is a member of the chat. Therefore an attacker can "iterate through all chats and learn display names of those users involved in private chats." Vanderbilt stated that from the display names an attacker can identify "the corresponding

usernames." Vanderbilt stated that this vulnerability "does not make use of a side channel in space."
Vanderbilt proposed two additional exploits which attempt to use a side channel in space. Vanderbilt concluded that because both exploits relied on their initial vulnerability they did not fit the definition of a Side Channel in space

Evaluation: Vanderbilt correctly concluded that there was not an intended SC Space vulnerability within the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.6.2 Question 004 (SC Space, Intended Not Vulnerable, Answered No)

Vanderbilt stated that their "tools for finding Side Channels are not currently able to handle cryptographic functions especially space-based Side Channels." They added that an idea for future tools would be to write a JVM "implementation or addon" that can "track different forms of managed" memory and follow how memory changes with requests.
Vanderbilt stated that the challenge question is "partially depended on the ModPow vulnerability which is not a [vulnerability] in space." "The server's private key is used only in the login phase" of the program. Vanderbilt limited their analysis to "products and co-products of the authentication" process. Vanderbilt noted that the authentication process "can be observed by looking at the com.cyberpointllc.stac.auth package, the KeyExchangeServer and KeyExchangeVerifier classes." Vanderbilt sampled the space by fixing the private key and varying the public keys and then fixing the public key and varying the private keys. Vanderbilt observed that keys are "divided into multiple equivalence classes by size" but "the number of keys in every equivalence class is too large […] to efficiently get the private key." Vanderbilt added that the SC space vulnerability leaks ≈2.06 bits of entropy for a ≈1035-bit key.

Evaluation: Vanderbilt correctly concluded that the application does not included an intended SC Space vulnerability. Vanderbilt claimed to have found an SC Space vulnerability which leaks ≈2.06 bits of entropy for a ≈1035-bit key. While this validity of this claim doesn't impact the post-engagement analysis ruling, Apogee may investigate to confirm the vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.6.3 Question 009 (SC Time, Intended Not Vulnerable, Answered No)

Vanderbilt noted that "to show a third party cannot identify an arbitrary search term, it is sufficient to show that there exists a class of search terms that a third party cannot disambiguate between." Vanderbilt "considered the class of all strings that are neither a special term not appear in any public thread of the database. These terms are not indexed by the database and return no hits when searched for." Vanderbilt ran their SPF Timing Side Channel Listener tool on a "simplified version of the code the show that all such terms are captured by the same path constraint – when the search term is neither a special term not in the database then neither the work done to retrieve the special text nor that to order and format the matching entries is performed." Vanderbilt stated that "all these terms map to the same timing observable meaning that they are in the same equivalence class with respect to that observable and cannot be differentiated between."

Vanderbilt analyzed the HashMap implementation within GabFeed and observed that the “data structure is used to index all the words in the application and is queried when the check for whether a search term is found in the database.” Vanderbilt was unable to analyze the HashMap implementation because their SPF Timing Channel Listener tool does not support hashing as a symbolic operation.
Vanderbilt noted that the HashMap implementation may cause searched for terms not found in the database to take differing amounts of time. This could potentially allow for them to be uniquely identified. Vanderbilt manually analyze the code and determine that the HashMap hashing each “string to one of N buckets with N starting at 16.” N doubles “whenever a certain number of collisions have occurred.” An SC Time vulnerability “may allow” an adversary “to determine which bucket a string has been hashed into if different buckets store differing numbers of strings.” Vanderbilt added that because the number of buckets is finite and “some may have the same size and infinitely many terms not in the database, there is some infinite subset of terms not in the database.” The SC Time vulnerability in the hashing function cannot be used to differentiate between the terms not in the database

Evaluation: Vanderbilt correctly identified that there was not an intended SC Time vulnerability within the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.6.4 Question 014 (AC Time, Intended Not Vulnerable, Answered No)
Vanderbilt repeated the AC Time analysis performed in the other variants of GabFeed and checked for the vulnerabilities present in other variants. Vanderbilt concluded that the challenge application did not contain an AC Time vulnerability.

Evaluation: Vanderbilt correctly concluded that the challenge application did not contain an AC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.7 Graph Analyzer

A.2.8.2.7.1 Question 013 (AC Space, Intended Not Vulnerable, Answered No)
Vanderbilt responded that “all of the indicated complexities are called exclusively by MinimizerBarnesHut, which relates to the BarnesHut algorithm. This is an approximation algorithm for performing an n-body simulation. It is recursive in nature with an order O (n log n), so unless the algorithm has been tampered with it should be well behaved. The nodes are stored in an octree, which is really a quadtree since the program only uses 2 dimensions and sets the 3rd to 0. Also, MinimizerBarnseHut is only implemented for a forcedirected layout. There are 3 layouts available via option selections: xy for simple plots where the user defines the x & y coordinates of the nodes, force for forcedirected layouts that uses weighting of the nodes and repulsion and attraction energies to determine the layout, and rand for random selection of the node coordinates. The random and xy plots exclude the suspicious Barnes Hut algorithm and for a given number of nodes and connections they tend to consume less resources. However, the forcedirected plots are more complex when there is a weighting factor associated with the nodes. This feature is apparently disabled, as it evokes a "Not Supported" exception, defaulting the weighting factor to 0 for all nodes and edges. Between the limitation of the weighting factor, the

dimensions being fixed at 2, and the practical limit of a maximum of around 35 nodes than can be defined within the 5000 byte input file size, it seems very unlikely that an input file can be constructed to cause the program to use more than 512 MB of memory to generate a graph. In testing, the maximum memory usage was < 70 kB for all valid input file sizes. In fact, even when an input file of 1024 nodes with 522753 edges was used (input file size 6,773,705 bytes), the max memory usage was still only 363904 bytes, where the limit is 1000x larger than this."

Evaluation: Vanderbilt correctly concluded that the challenge application did not contain an AC Space (memory utilization) vulnerably.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.7.2 Question 020 (AC Time, Intended Vulnerable, Answered No)

Vanderbilt responded that "the getRepulsionEnergy method seemed a likely candidate in that it performs a recursive call on itself. However, the recursion is limited to the number of child nodes and given that the input dot file has a limit of 5000 characters maximum limits the total number of nodes that can be defined to a maximum of around 35, with each node connected to every other. The methods that were highlighted as the potential spatial complexity weakness are all part of the BarnesHut Minimizer algorithm that is used when " *force* " is used as the 3rd argument to the GraphAnalyzer executable to generate a force directed layout (the other options being " *xy* " for a user specified layout and " *rand* " for a random layout). However, in order for a force directed layout to have nodes with a weighting factor other than 0, the dot file parser must support defining the nodes with a weighting factor. Doing this causes a "Not supported" exception to be thrown, so the force directed layouts are all assigned a weight of 0, which limits the bounds of the calculations in the algorithm. The greatest complexity in both space and time was in defining nodes that fall in an evenly spaced circular pattern with all nodes connected to each other. With the input file limitation of 5000 bytes, the maximum RSS memory usage was around 66 kB, with a maximum execution time of 8 sec and an output file size of 438 kB. A variety of node configurations have been attempted, using all 3 (rand, xy, force) layouts. *rand* produces the smallest time/ memory/ file size footprint and *force* the largest (albeit only in time complexity compared to *xy*).

Although obvious, it should be mentioned that the greater the number of nodes (up to a point) and the greater the number of non-overlapping connections, the greater the output file size and time of execution. Additionally, various node patterns seem to indicate that a circle generates the greatest time/size characteristics for a given number of nodes. […] Although Acme.Inthashtable seems like it could be vulnerable by exploiting collisions to make a long chain, the Janalyzer shows that this piece of code is unreachable, which was verified by inspection of the source code. Thus, it cannot be a vulnerability."

Evaluation: Vanderbilt correctly noted that the greater the number of nodes and the greater the number of non-overlapping connections, the greater the output file size and time of execution. Vanderbilt was incorrect in stating that the limit of 5000 characters limits the total number of defined nodes to 35. Vanderbilt missed the vulnerability in the challenge program which allows users to define subgraphs from previously defined graphs with the "container:" keyword. This significantly increases the number of nodes that can be defined using the 5000 character limit. When the "PNG" output format is selected the resource usage limit can be exceeded.

Post-Engagement Analysis Ruling: Incorrect

A.2.8.2.7.3 Question 034 (AC Space, Intended Vulnerable, Answered No)

Vanderbilt used their Janalyzer tool to identify "6 occurrences of 7-level nested Loops and 32 occurrences of 6-level nested loops. All of the 7-level loops involved minimizeEnergy and either getRepulsionEnergy or getAttractionEnergy, which did not seem to present any vulnerabilities. However there were also 2 occurrences of loops involving saveGraphics, paint, paintContents and ending in writeSetDash in the 6-level loops. These are also involved in the generation of the output files, which could be a potential vulnerability.

After inspection of the code involved, Vanderbilt identified several nested loops in the calls. Vanderbilt noted that the nesting tends to be deeper for PS output files than it does with PNG, but in all cases the loops seem to be quite well behaved. There are no cases where a loop parameter is being modified within the loop, and the loop limits seem to all be reasonable. For instance, base.CmdSavePS.saveGraphicsLayer iterates over each Fig entry in the graphic to draw, as does base.Layer.paintContents and base.LayerManager.paint iterates over the number of layers to paint. These are all well-defined loops and as such behave in a linear fashion based on the number of lines that need to be drawn, which in turn is based on the number of nodes and edges that are defined.

In the testing that was performed the greatest file sizes are produced when the nodes are arranged in a circular pattern and all nodes are connected to each other with as few overlapping lines as possible. When this was done, the largest output files that were produced for an input file that was < 5000 bytes in size was 319 KB for a PNG file type and 20 KB for a PS file. If the input file size was disregarded to some extent," Vanderbilt stated that they "can generate a circle consisting of 100 nodes with an input file size slightly over 50 KB and the PNG file produced is 428 KB and the PS file size is 253 KB, still 1000x less than the challenge size.

Interestingly, if double and triple concentric circles are generated (with the angle skewed a bit on each circle to minimize the lines that overlap), as the nodes increase to 300 (with input file size > 500 KB), the PNG file size output actually drops to around 32 KB, but the PS file size increases to > 2 MB. In both cases, though the execution time increased by about the same amounts. […] Graph Analyzer does not seem to have a space complexity related to the size of the output file produced."

Evaluation: Vanderbilt did not identify the intended AC Space vulnerability in the application. Users can define subgraphs from previously defined graphs with the "container:" keyword. This significantly increases the number of nodes that can be defined using the 5000 character limit. When the "PS" output format is selected the resource usage limit can be exceeded (the "PNG" output format limits output files to 2000x2000 pixels). Vanderbilt made an interesting observation that after inspecting the code that generates outputs the loop nesting was deeper for PS output files that for PNG output files.

Post-Engagement Analysis Ruling: Incorrect

A.2.8.2.8 Image Processor

A.2.8.2.8.1 Question 005 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt first ran the VisualVM profiler tool and observed that the application spends a lot of time in the *exp(x,n)* method. Vanderbilt's Janalyzer tool reported that the *exp(x,n)* method

contains 5-level deep loops. Vanderbilt rand their WorstCaseAnalyzer tool on the *exp(x,n)* method and noted that "the method is linear with respect to the input parameter n." Vanderbilt constructed a driver for the Intensify filer. Vanderbilt analyzed the input with "the TimingChannelListener to find the worst case path for a single input size." The tool showed that "the worst-case pixel is one with the integer value of 30."
Vanderbilt concluded that the application was vulnerable to an AC Time vulnerability. "The Intensify filter contains a call to com.stac.mathematics.intensify() , which applies some transformations to a pixel. It is called 3 times per pixel. The method looks up a value in a static array at location r+g+b%255, where r, g and b are the red, green and blue values of the pixel, respectively. It then calls the method exp() with the value from the array as parameter. That value is used as the bound for a loop inside that method. The worst case execution therefore corresponds to the maximal value in that array, which is the value at index 30." Vanderbilt confirmed the vulnerability by constructing an image which caused the application to exceed the resource usage limit.

Evaluation: Vanderbilt correctly identified the vulnerability within the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.8.2 Question 030 (AC Space, Intended Not Vulnerable, Answered No)

Vanderbilt used a worst case analyzer that used accumulated heap space as a cost model. This model currently only accounts for allocations and "does not track live objects". This causes the model to "over-approximate heap space size". Vanderbilt used this model to verify that the "'detect' method which performs a number of transformations on BufferedImage instances" was not vulnerable. The worst case analyzer assumed an "input parameter that denotes the input size". For analysis Vanderbilt used the size of a square input image. To analyze the method Vanderbilt used input sizes of "1 x 1 pixel to 200 x 200 pixels" and a "step size of 20 [pixels]". Vanderbilt obtained a quadratic fit function of: $16251.06 - 395.41x + 97.55x^2$. Vanderbilt used this function to "extrapolate the heap space for the maximum image size of 500 x 500 pixels." This resulted in a value of ~23 MB for the size of the heap. Vanderbilt performed manual analysis using the Memory Analyzer (eclipse plugin) and only observed a ~3.7 MB heap size. Vanderbilt concluded that while the worst case analyzer model was an over-approximation of heap size the value produces was still below the resource usage limit.

Vanderbilt conducted further analysis and applied their "Janalyzer tool to find memory allocations inside nested loops." The tool identified a 4-level deep loop in the com.stac.image.algorithms.generics.Canny EdgeDetect.detect() method. Vanderbilt observed that the "method allocates 8 new BufferedImage instances." The Janalyzer tool identified the 'detect' methods for other filters as potentially vulnerable.

Vanderbilt noted that after running a diff of the Engagement 2 application with the Engagement 1 application the main difference was an introduced limit of 250,000 pixels on an input image.

Vanderbilt analyzed the CannyEdgeDectect.detect() method and observed that it "performs edge detection by calling method in java.awt – in particular it calls java.awt.image.Convolve.convolve which creates a copy of the input image. The [convolve] method is called several times along with other methods. When the detect method is about to return, it has a total of 9 BufferedImage instances. The size of a BufferedImage instance is determined by the dimensions of the image and the image type." The method "keeps 2 references to the input image (one for the original the

other for a blurred version of the original) and 7 references to BufferedImage instances with the same dimensions as the input image but with the GRAY, 8-bit, image type."

Vanderbilt concluded that the application does not seem to have an AC Space vulnerability.

Evaluation: Vanderbilt correctly concluded that the application does not contain an intended AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.9 Law Enforcement Database

A.2.8.2.9.1 Question 019 (AC Time, Intended Not Vulnerable, Answered No)

Vanderbilt responded that the interaction with the challenge application has two components: "a logging functionality [… and] interaction with the backing store." The logging functionality "logs client requests to file(s). This is carried out by […] communication over a socket that connect on port 6666. The server socket writes the contents to file. The data received is of a constant size (inserted key, range, etc.)" The "backing store […] is a b-tree of order s=10 (minimum of 10 keys in a node. A node can have a maximum of 2*s -1 keys)." Vanderbilt used "m = 2*s – 1 in the following" analysis.

The b-tree limits each node to 20 children (and therefore 19 keys). "Every non-leaf node (disregarding root) has ceil(20/2) children. A non-leaf node with k children has k – 1 keys stored. All leaves are on the same level (balanced). Insert operations are O(log n) where n is the number of elements. Searching for an element is O(log n). A range search of [min; max] corresponds to a traversal of the tree delimited by searching for min, and a traversal until some key greater than max is found. The worst case height of the b-tree is floor(log(n+1/2)) base ceil (20/2) – the root has a height of 0. Vanderbilt noted that the worst-case configuration of a b-tree happens when the nodes are half full. When INSERT is performed, no satellite data is associated with the key. When the tree is initialized, satellite data is associated with the key (it is the number appearing after ";" in the dump file, which is the same as the key). The determination of restricted IDs is done independently – this information is stored in an arraylist."

The logging functionality appeared "to take constant time." The time "is independent of the actual data sent. It simply logs the requests." Vanderbilt added that "if a vulnerability is present it must be" in the interaction with the backing store. Vanderbilt's "intuition of the worst case behavior was to perform a number of INSERT operations followed by a SEARCH operation. A SEARCH doesn't seem to alter the state of the server. Given the budget' Vanderbilt observed that "198 insertions" can be performed, "thus the b-tree can be configured with a total of 529 keys. The subsequent SEARCH operation should span over the entire tree." Vanderbilt added that "given 529 inserts the tree will at most consist of 3 levels." Vanderbilt concluded that "making an INSERT operation the last operation is unlikely to trigger worst-case behavior because only a few nodes need to be split in the worst case."

Vanderbilt "ran the worst case analysis with 529 symbolic insertions" and performed a SEARCH operation after the insertions. Vanderbilt used a "heuristic search for input size 529." Vanderbilt generated a graph of input size vs tree depth.

Vanderbilt concluded that the challenge application did not contain an AC Time vulnerability

Evaluation: Vanderbilt correctly concluded that the challenge application does not include an intended AC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.9.2 Question 026 (SC Time, Intended Vulnerable, Answered Yes)

Vanderbilt used code inspection to identify the "SEARCH and INSERT operation […] as the probable" sources of an SC Time vulnerability. Vanderbilt wrote a Symbolic Pathfinder (SPF) "driver which inserted two unrestricted ID's and one restricted ID into the database." Vanderbilt "used symbolic execution with the TimingListener class to determine the execution time for different execution paths." Vanderbilt analyzed the SPF output and saw that "the SEARCH operation takes longer when the secret is within the search range (9059, 8713, 8701 byte code instructions) as opposed to the case when the secret is out of the search range (7916, 7951 byte code instructions)." Vanderbilt proposed and tested a binary search on the range of possible keys (0 to 40,000,000). Vanderbilt observed that this attack worked because it used timing on the server-side. On the client-side there is too much noise for the attack to work.

Vanderbilt "observed that the SEARCH operation sends the unrestricted keys found within the range one-at-a-time, in separate UDP packages, after each key is found. When there is a secret key between two consecutive unrestricted keys, the delay between two messages will be longer due to the timing side-channel described above." Vanderbilt used this method to identify the gap containing the secret and run a binary search within the gap.

Evaluation: Vanderbilt correctly identified the intended SC Time vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.9.3 Question 027 (SC Space, Intended Not Vulnerable, Answered No)

Vanderbilt responded that "the only observable space side channel is the size of the UDP packages at the client-side. This size corresponds to the allocated byte buffers sent from the server (class UDPServerHandler, channelRead0 method, case 8). Using code inspection, […Vanderbilt] identified that the size of the UDP packages correspond to the allocated byte buffers sent from the server in the SEARCH query (class UDPServerHandler, method channelRead0, cases 8) and are the probable source of the side channel.

Vanderbilt "wrote an SPF driver which inserts five concrete unrestricted IDs and one symbolic restricted ID into the database. […They] used symbolic execution with the SpaceSideChannelListener class, to determine the size of the packages sent for different execution paths. […Vanderbilt] observed that each package sent from the server-side and received on the client is 4 bytes large. […Vanderbilt] then observed a side channel due to an IndexOutOfBoundsException thrown within the SEARCH query. When the restricted key is found within the search range, but is after the last unrestricted key in said range the applications throws a RestrictedKeyFoundException this results in an allocation of a 1 byte package.

In order to observe if this package is sent at runtime, which could potentially lead to a side channel attack, […Vanderbilt] conducted the following analysis with SPF. […Vanderbilt]

changed all the byte allocations (ByteBufs in the original code) to dummy method calls. [.. They] mocked the 4-byte allocations by an Allocations.FOUR_BYTES method, and the 1-byte allocation by an Allocations.ONE_BYTE method. These methods are empty. [… Vanderbilt] wrote a space side channel listener which traced the calls to the dummy methods they implanted into the application, and kept track of the number of packages sent, and the total size of each of those packages.
At the end of the execution, [… Vanderbilt] checked if the sum of the sizes of UDP packages sent divided by the number of packages sent was exactly 4. If not, the application included execution paths where the exception described above is triggered, allowing leakage through the space side channel.
[… Vanderbilt] noted that the probability of triggering the space side channel depends on the number of non-restricted keys in the database." Vanderbilt created a table which "shows how this probability changes with an increasing number of non-restricted keys, for different distributions of non-restricted keys."
Vanderbilt concluded that "given the operational budget of 50, the space side channel is not sufficient to be exploited. In general, to exploit this side channel, [… Vanderbilt calculated that they] would need n + log(r) operations, where n is the number of unrestricted keys in the database, and r is the size of the space between two public keys enclosing the restricted key (which [an attacker] can explore in log(r) operations, using binary search."

Evaluation: Vanderbilt correctly concluded that the application did not contain an SC Space vulnerability that could determine a secret key within 50 operations. Vanderbilt claimed to have discovered an unintended SC Space vulnerability which did not meet the operational budget. Apogee will investigate to confirm this vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.10 SnapBuddy 1

A.2.8.2.10.1 Question 012 (SC Time, Intended Not Vulnerable, Answered No)

Vanderbilt "used Wireshark to gather timing and space data from packets sent and received" by the application. Vanderbilt's use of Wireshark "was meant only as a tool to aid with manual inspection." Vanderbilt did not identify an SC Time vulnerability using Wireshark.
Vanderbilt conducted further analysis using their SPF WorstCaseAnalyzer and their Fuzzer tool. Vanderbilt repeated the analysis performed for question 25 for AC Space. In question 25 Vanderbilt identified that this variant of the application contained a sort vulnerability which causes the application to take "longer when the number of items to be sorted is a multiple of 3". Vanderbilt noted "that the only place where sorting occurs is when Friends or FriendsPhotos is displayed, or when looking for nearby neighbors."
Vanderbilt observed that the "number of users in the application is 448. If a single user were to have nearly all users as friends and the user's total number of friends were divisible by 3, there would be a significant spike in timing when Friends of Friends Photos are displayed. If an attacker could get a baseline timing for a user with no friends and one with all friends, the attacker could determine that the user in question has everyone as friend."
Vanderbilt stated that "aside from the above scenario [… they] were unable to identify any" other SC Time leakage within the application. Vanderbilt added that they "are still unable to prove the absence of any SC Time vulnerability with [… their] current tools."

Evaluation: Vanderbilt correctly concluded that there was not and intended SC Time vulnerability within the challenge program.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.10.2 Question 025 (AC Space, Intended Vulnerable, Answered No)

Vanderbilt ran their Janalyzer tool and identified two areas for further exploration. The first area was the "teeifyHelper, resizeHelper, and treeify in the hashmap implementation." The second area was the filter methods which "can allocate several BufferedImages". Vanderbilt noted that it may be possible to "upload many images and apply several filters" to each "to trigger a memory vulnerability". Vanderbilt observed that "a maximum of four filters can be applied to a photo." Vanderbilt conducted further analysis and observed that they were "able to achieve 350 kB" of RSS memory usage. Vanderbilt proceeded to analyze the sorting method in the application. Vanderbilt observed that the sort method has a different behavior when the length of the list is a multiple of 3. Vanderbilt was able to confirm an AC Time vulnerability in the sort method. Vanderbilt used their SPF WorstCaseAnalyzer to determine that "sorting a semi-random list of length 3000 consumes 524,936 kB of memory" which exceeds the resource usage limit. Vanderbilt proceeded to analyze the reachability of the sort method and concluded that the method is always "called on *Person* objects which are sorted in ascending order by name." The sorting method is "called from *FriendsHandler*, *NeighborsHandler*, and *FriendsPhotosHandler*." Vanderbilt concluded that the "vulnerability could be triggered if they could" get a person K friends or neighbors "where K is a large enough multiple of 3". Vanderbilt used their fuzzer to test the response time of the server to various sort method inputs. Vanderbilt confirmed an AC Time vulnerability in the application with sort input lengths which are divisible by 3.

Vanderbilt noted that in SnapBuddy Friends and Neighbors are sorted. With 448 users defined in the system, there are a "maximum of 447 Friends or Neighbors". All user begin with "four friends and the only way to add friends is to send invites and receive responses." Vanderbilt exceeded the input budget and used >200 KB to add all users as friends of a single user. Vanderbilt observed that displaying the Friends and Friends Photos still consumed less than 250 KB.

Vanderbilt concluded that the application contains an AC Space vulnerability but the user would have to have >3000 friends for to exceed the resource usage limit.

Evaluation: In SnapBuddy users can add filters to their photos. Each filter operation uses memory; in order to prevent users from exceeding the resource usage limit, there are a limited number of filters which users can use and users cannot apply duplicate filters. The duplicate filter check for the 'ScaleUp' filter is case sensitive, an attacker can specify the same filter multiple time by using a different combination of cases for each letter in the command. Using this method the attacker can exceed the resource usage limit. Vanderbilt did not identify the intended AC Space vulnerability but identified a potential AC Space vulnerability which could not sufficiently meet the input budget limit and exceed the resource usage limit

Post-Engagement Analysis Ruling: Incorrect

A.2.8.2.10.3 Question 031 (SC Space, Intended Vulnerable, Answered Yes)

Vanderbilt used their Janalyzer tool to "identify location-related method calls within the application." Vanderbilt observed that "a user has to set his location immediately after logging in if his location is not previously set. A user can optionally update his location immediately after he logs in. A user can update his/her location by navigating to https://localhost:8080/location after" logging in. "A user has to input a list of BSSIDs as a location identifier to update his location". Vanderbilt used manual crawling and code inspection to identify "that one has to put an exact number of BSSIDs to update [one's] location. Vanderbilt visited "https://localhost:8080/cities" to identify all cities and their corresponding BSSIDs. Vanderbilt analyzed the BSSIDs and observed that each city maps to a "unique set of BSSIDs of a unique cardinality. In order to set/update your location you have to provide the exact set of BSSIDs for the desired location. As a result, each location set/update request has a unique encrypted package size based on the number of BSSIDs that describe the desired location." Vanderbilt added that "an attacker can view a user's entire session and can log the sizes of unencrypted packages. Since the encryption is size preserving, the attacker can figure out the location of a user by mapping the request package sizes to the corresponding locations. Note that there may be other packages which may have the same size as a package that corresponds to a location update request. An attacker can filter out that packages based on the order of request/response packages in a complete session."
Vanderbilt concluded that "one cost can be used by an attacker to learn the information about the sizes of encrypted packages corresponding to BSSIDs of each location in the app, one cost for the attacker to observe encrypted requests/responses for a user, and one cost for an oracle query to check if the user is in the given location."

Evaluation: Vanderbilt correctly identified the indented vulnerability within the challenge program.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.11 SnapBuddy 2

A.2.8.2.11.1 Question 002 (SC Space, Intended Not Vulnerable, Answered No)

Vanderbilt responded that "the source code was inspected using Janalyzer, especially on those parts related to user authentication; impersonation could be done by stealing a user's password or session key. The application was manually crawled and the network packages during user login/logout actions were observed. No side channel relating to size was identified."
Vanderbilt added that "SSL uses block encryption. An HTTP request and response is encrypted together with its necessary headers and body in the network layer. If [an attacker] can identify HTTP requests or responses that are encrypted in a way that the total header size is a multiple of the block size that is used by the encryption algorithm used, [the attacker] can manipulate the body of the encrypted message under certain conditions depending on the encryption algorithm. That allows [the attacker] to change the body of certain http requests or HTTP responses which can further allow [the attacker] to impersonate a user. In that case a space side channel would be the sizes of the network packages (HTTP requests or HTTP responses) and an attacker maps package sizes to HTTP requests or HTTP responses that can be manipulated." Vanderbilt was "not able to verify the existence of such an attack."

Evaluation: Vanderbilt correctly responded that the application does not contain an intended SC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.11.2 Question 015 (SC Time, Intended Vulnerable, Answered Yes)

Vanderbilt "found a timing channel vulnerability in the method *modpow* of the class *ModPow*. This vulnerability reveals information about the private key used in the server's authentication. By manual inspection of the modular exponentiation function, [Vanderbilt saw] that there is a branch that is only taken when the i^{th} bit of the secret key is 1. Thus, [Vanderbilt] believed there is a side channel which reveals the number of 1 bits in the secret. [Vanderbilt] performed a sample-based profiling of the modular exponentiation functions found in all versions of the GabFeed problems in which they varied the number of 1s in the secret and recorded the computation time." Vanderbilt observed that there is "little noise that obscures the channel, and that execution time is correlated with the number of 1 bits in the exponent". Vanderbilt concluded that the side channel was weak.

Vanderbilt added that they are aware that "there is a well-known timing attack which does reveal the bits of the secret, one at a time, due to multiplication optimizations. This attack works by measuring the times between two specially chosen base values and using them to perform a binary search on the smaller prime factor of the modulus. Knowing the prime factorization is enough information the recover the secret exponent. This attack is described in *Remote Timing Attacks are Practical* by David Brumley and Dan Boneh. [Vanderbilt] attempted to implement this attack, but were not able to make it work, even when they removed the programmatic noise and isolated the timing to be only on the server without the network communication."

Vanderbilt added that currently their "tools are not able to handle this problem due to a few limitations. First, [they] have limited techniques for analyzing *modpow* using symbolic execution for very small key sizes (6 bits). However, symbolic execution does not scale to analyzing programs with realistic key sizes.

Second, the constraints that [they] derive from ModPow using symbolic execution are nonlinear, and so their model counting techniques using LattE and ABC do not apply to those constraints."

Vanderbilt stated that "there are two areas for improvement: be able to handle larger key sizes, perhaps by making only a few bits of the secret symbolic at a time, and incorporate model counters for nonlinear constraints, or for BigInteger / BitVector constraints."

Vanderbilt concluded that "there is a timing channel vulnerability". They believe that "it can reveal the secret within the budget, but the current implementation of their tool can't verify the hypothesis."

Evaluation: The challenge application contained an SC Time vulnerability due to two sets of conditionals in the application. In the ModPow.modPow() method there is an if statement where the conditional is dependent on the value of each individual bit of the server's private key. Within the if statement there is a call to the OptimizedMultiplier.fastMultiply() method. This method contains an "if, else if, else" block where the conditional is dependent on the values of the variable "s", and the variable "base" which is the value submitted by an attacker. The side channel which leaks the server's private key is dependent on the differential processing times within the ModPow.modPow method and the OptimizedMultiplier.fastMultiply() method. Vanderbilt stated in their response that they identified a vulnerability in the modPow method but correctly concluded that it was not sufficiently strong. Vanderbilt added that they were aware of

well known timing attacks which reveal the bits of the secret one bit at time. Vanderbilt did not identify the intended SC Time vulnerability present in the challenge application. The exploit provided by Vanderbilt does not satisfy the operational budget.

Post-Engagement Analysis Ruling: Incorrect

A.2.8.2.11.3 Question 028 (SC Space, Intended Vulnerable, Answered Yes)

Vanderbilt's Janalyzer provided a quick visual summary of the application code. They launched the application and crawled it manually to identify any interaction that is related to a user's friends.

Vanderbilt assumed "the attacker has an account on the system and can view all the profile pictures along with corresponding user names and ids (all profile pictures are public via/thumb/<userid>/profile.jpg)."

"The attacker can collect all profile pictures using the *https://localhost:8080/invite* URL along with user names and user IDs. Profile pictures are in JPEG format and they mostly likely have different sizes. In the current application status there are 448 profile pictures and 445 of them have unique file sizes. File sizes vary between 86812 bytes and 287644 bytes. There may be fewer collisions even though there are thousands of users in the application. The attacker can also upload a profile picture with a unique size.

The attacker then listens a user's network traffic through a session. When a user requests a page with images, browser requests image resources as a separate request. The attacker can differentiate friend/unfriend cases considering his profile photo. If the attacker knows the location of the user, he can also identify the case where the user requests neighbors by putting himself into the same location as the user. By also examining the flow of the requests and number of profile photo request match (based on size) the attacker can divide the above set of URLs into two: 1) friend related 2) non-friend related. By doing so he can identify at least one friend of the target user.

The attacker can also identify the different URLs based on the referral entry in the HTTP header. For example, when a user requests *https://localhost:8080/friends*, the response includes HTML that includes img tags for profile photos. When a browser sends get requests for each img tag, it includes a referral link header which corresponds to the https://localhost:8080/friends. Each URL above has a different length, as a result encrypted HTTPS requests for an image will have different sizes based on the referral link. The attacker can map <request, response> tuples to <header size, photo size> tuples to identify both the requested page and the image resource loaded for this page."

Vanderbilt" used the Wireshark network sniffing tool to validate above observations. Based on the above attack, the attacker can identify at least a friend of a target user."

Vanderbilt concluded that "there is a side channel in space which allows an attacker to identify one or more friends of a user by observing its communication channel given the operational budget."

Evaluation: Vanderbilt correctly identified the intended vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.11.4 Question 035 (AC Space, Intended Not Vulnerable, Answered No)

"Given that this question asks about a space vulnerability", Vanderbilt used their Janalyzer tool to "find potential points at which excessive 'new' memory allocations occurred (and can keep occurring, resulting in exceeding the resource usage limit)". Vanderbilt also used their SPF WorstCaseAnalyzer to "analyze some critical algorithms in the program".

The Janalyzer tool reported many inter-procedural loops ranging from a depth of 1 to 6. The Janalyzer tools also reported "'new' allocations that take place inside inter-procedural nested loops." Vanderbilt identified 6 areas of the code that could contain an SC Space vulnerability. Vanderbilt began by analyzing the HashMap class. Vanderbilt observed that "the put method recurses and allocates new Node and AbstractMap.SimpleEntry objects. The treeifyHelper method creates new TreeNode objects and is called in many situations. The resize method allocates memory for new Node objects. The implementation looks fine." Vanderbilt concluded that the HashMap class was not vulnerable because the program "stays within resource bounds, even with the large file that causes collisions in the HashMap". Next Vanderbilt analyzed the Tree Node class. The "putTreeVal method potentially creates new TreeNode objects" but Vanderbilt concluded that the class was not vulnerable. In the LocationServiceImpl class, "the constructor requires some user input, which can ultimately trigger the byteArrayToHexString method. Vanderbilt noted that they did not think this was vulnerable, but with enough input, this could be overloaded." Vanderbilt added that the UserManager class contained an "addUser method that could potentially be overloaded". Vanderbilt observed that "many Handler functions can be overloaded because they call on the HashMap class (put > putHelper > treeify > treeifyHelper). Specific Handler functions include InviteHandler.getContents, FriendsPhotoHandler.getContents (this one is especially threatening, as it is a part of many control flow paths present in the Janalyzer), and PhotosHandler.getContents". Vanderbilt ran their SPF WorstCaseAnalyzer on the sorting algorithm and showed that its memory usage was linear with respect to the size of the list. They concluded that there was "probably no vulnerability related to the sorting algorithm". Lastly, Vanderbilt ran their Fuzzer tool using "randomly generated strings and found no evidence of a space complexity vulnerability". Vanderbilt concluded that they" did not find a space vulnerability in this program". However, their "tools are currently limited in their ability to perform automated analysis to find memory allocation vulnerabilities." Vanderbilt added that "one idea for future automation is to allow the user to select memory allocation locations reported by the Janalyzer and automatically generate "hooks" that are called via an attached agent that generate a heap dump after those memory allocations are performed. Then, this generated heap dump could be given as input to a memory analysis tool, such as the Eclipse Memory Analyzer, to analyze for memory vulnerabilities."

Evaluation: Vanderbilt correctly concluded that the challenge application did not include an intended AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.12 SnapBuddy 3

A.2.8.2.12.1 Question 018 (SC Time, Intended Not Vulnerable, Answered Yes)

Vanderbilt observed that a way to "achieve the goal of impersonating a given user is to discover their password and simply log in with their information. To determine whether such an attack is possible, Vanderbilt started by running SPF on the password matching method found in the User

class. In each iteration of the for-loop, two characters are compared. [Vanderbilt's] analysis with SPF showed that the path taken each time these two characters are not equal is one instruction longer than the one taken otherwise. This extra instruction is simply the jump instruction taken at the end of the "*if*" branch. This results in a side channel in time which, if the timing measurements are refined enough, one can tell how many characters were matched, though it reveals no information about the position of the character in the encrypted password."

Vanderbilt "experimented with the DES encryption method found in SnapBuddy and, as in GabFeed, and found two important features. The first is that passwords are encrypted in blocks of size 8 with every eight characters being encrypted independently with the only additional consideration being their offset from the start. This means that if one can discover how to break the first 8 characters of any password, breaking the rest is just iteration on the same idea. The second feature is that any set of characters of the same length with length less than 8 are encrypted to the same thing. This means that one can break any password of length less than 8 and also that the attacker only need to break the password length % 8 number of blocks for any password.

If passwords of length one were encrypted to matching hashes of length one, then a simple attack would be find a password that encrypts to 'a', determine if 'a' is the first character using the side channel in time given above and then move on to a password that encrypts to 'b' and so on. This would allow the attacker to determine a password in (alphabet size) * (length of encrypted password) time. However, there are two problems with this method. First, as described above, all characters of the same length less than 8 encrypted to the same string. This means the attacker must begin working on password of size 8 as the smallest case, beginning with 'axxxxxxxxx=' (passwords of length 8 encrypt to strings of length 11 plus an extra padding '=' symbol), then 'bxxxxxxxxxx=' and so forth. Second, the attacker is comparing encrypted passwords, not the passwords themselves. This means one must find a way to go from the desired hashed password to the input that would hash to that password. Basically, the attacker is looking for an inverse.

DES is not a hash but rather an encryption method. Moreover, the Feistel algorithm used to encrypt has some nice properties. It is a symmetric cipher, meaning the same key is used for both encryption and decryption. Decryption in our case is simply iterating through the rounds of the algorithm using the subkeys in reverse order. Thus the problem of finding an inverse reduces to the problem of finding the key used in the DES encryption.

There are several potential methods to do this. One side channel attack against DES uses how the time taken by the permutation step of the algorithm is generally dependent on the Hamming weight, or number of ones, of its input. In our case, the code used during the permutation step avoids explicitly branching based on if the considered bit is a one or a zero by doing a bitwise shift followed by an "and" operation: *this.dst = (this.dst << 1 | (this.src >> srcPos &0x1L)*.

While looking at the instruction level code for this class revealed that there were no explicit branches, there is a very high possibility that the time taken when the considered bit of this.src is a 1 is larger than when it is a 0 depending on the optimizations done by JIT compilation. This is because of the need to set a bit to 1 as opposed to merely leaving it as a 0. If this is assumed to be the case, then there is a side channel in time that can be used to reveal the secret key. It works as follows. The permute function is called at the end of each call to the Feistel function. 4 bits of the

input to this permutation function are determined by 6 bits of the subkey being used in the current round (along with bits from the input message). There are 64 possible values for the 6 bits of this subkey. Using this information, the attacker can compute a time model for each of the 64 values based on the Hamming weight of the 4 bits of input to the permute function it results in. This time model is based on a sizable number of chosen plaintexts. The attacker then attempt to log in using the chosen plaintexts and observe the resulting times. For each of our 64 time models, the attacker determines its Pearson correlation coefficient with the observed output and chooses the one with the highest correlation. This allows the attacker to determine those 6 bits of one subkey and can proceed iteratively. For a step by step explanation of this attack, see: https://cybermashup.com/2013/12/13/timingattackspart1.

This attack requires the time side channel in the permute function in order to work. In fact, running some profiling to determine whether the time taken in the permute function is proportional to the Hamming weight of the input seemed to reveal that there was no correlation. As such Vanderbilt did not feel confident that this attack can be used. Even without this side channel though, there is another side channel that allows an attacker to extract the key used in the encryption. This is a side channel in time that uses cache memory to determine the secret key. This attack is based on the time taken to index into an Sbox being related as to whether or not that index was recently used. If the index had been used recently, the result would be stored in the cache and would be faster to retrieve. Since the index into the Sbox is dependent of the bits of the subkeys, this can leak information about the values of those bits, allowing the secret key to be obtained. For a more detailed explanation of how such an attack would work, see: https://eprint.iacr.org/2002/169.pdf, https://cr.yp.to/antiforgery/cachetiming20050414. pdf, https://www.rocq.inria.fr/secret/Anne.Canteaut/Publications/RR5881.pdf"

Vanderbilt concluded that "using one of the methods outlined above, it should be possible to retrieve the password key. Using this password key, it is possible to invert the DES encryption allowing an attacker to gain the required input to execute the previously outlined timing attack. What remains to be determined is whether this attack is possible in the allowed budget. Since active operations have no cost, it is a simple matter to attempt to log in with enough plaintext to generate the timing information needed to find each bit/six bits of the DES key per one session a piece. Likewise, the attacker can use one user session to collect timings for the inverses of 'axxxxxxxxxx=','bxxxxxxxxxx=' and so on and hence can crack one character per session. This allows the attacker to break the password well within the budget of 10,000 user sessions."

Evaluation: Vanderbilt reported an unintended vulnerability which is claimed to allow a third party to learn information that allows the third party to impersonate any given user. Impersonating a user requires identifying the user's login credentials (username and password). Vanderbilt did not provide a method by which an attacker can determine a user's username. Vanderbilt stated that the password matching method found in the User class takes different paths based on whether two characters being compared differ. Vanderbilt added that the path taken each time the two characters are not equal is one instruction longer than the one take otherwise. The method doIt3 in private class ClasspasswordsEqual within the com.cyberpointllc.stac.webserver.User class is used to verify a user's password. The method is shown below:

```
min = Math.min(aLen, bLen);
for ( int i = 0; i < min; i++) {
```

```
            if (a.charAt(i) != b.charAt(i)){
                    equal = false;
            } else {
                    shmequal = true;
            }
    }
    return equal;
```

The method iterates though the shorter of the two inputs (a valid user's password, and the submitted password). The runtime difference between two equal strings and two strings which differ by one character was observed at 5 nanoseconds. This timing would have to be completed on the server side. Given the noise within the entire application running over localhost on the reference platform, the EL does not believe that this side channel is strong enough to provide a 95% probability of success.

Post-Engagement Analysis Ruling: Incorrect

A.2.8.2.12.2 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt used Janalyzer to recognize that "many filters have deeply nested loops". Vanderbilt tested each available filter and identified the OilFilter as potentially vulnerable. Vanderbilt used the ScaleUp (F00E) and the 5 longest running filters (OilPainting(F00F), Chrome(F00A), Crystallize(F00B), GaussianBlur (F013, Kaleidoscope (F007)) and observed a response time of 379 seconds on the reference platform.

Vanderbilt also observed that SnapBuddy 3 does not have a guard against duplicated filters and does not limit the number of filters added to an image. The web UI limits the number of filters at 6 but the server side service does not check. (Note: these checks are in place in SnapBuddy 1 and 2.)

Evaluation: Vanderbilt identified the intended vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.13 SubSpace

A.2.8.2.13.1 Question 041 (SC Time, Intended Not Vulnerable, Answered Vulnerable)

Vanderbilt used their "Symbolic PathFinder (SPF) tool with the side channel listener on the *isEqual* method of the *Crypto* class, the *Search* and *Add* methods of the *GeoMultiTrie* and the *registerUser* method of the *Database* class." Vanderbilt found three SC Time vulnerabilities that leak information about the secret.

"The method *isEqua*l of the class Crypto is used in authentication. The timing channel leaks information about the encrypted password of a user. SPF determines that the channel capacity of the leak is just 1 bit, not enough to reveal all the password. After a user A sends hail, the server will look for A's nearest neighbor. The locations of all users are stored in the GeoMultiTrie data structure, which is also responsible for finding nearest neighbor. The discrepancies in the response times of searching on this tree leaks information about the location of a user. Since Subspace is an anonymous message, this is enough to impersonate a user. [Vanderbilt] attempted to run SPF on the Search and Add, however, these two methods makes calls to native methods longBitsToDouble and doubleToRawLongBits, so they were not able to reach a conclusion on this timing channel.

When the user registers a new account, the server will check if the username and email provided by the user already exist. This check includes a lookup in the hashmap usernameToUser and emailToUsername. Knowing the username of a user is enough to fake a “hail" from this user (impersonation). Vanderbilt ran SPF on the method *registerUser* of the class *Database*, and the results show that the channel capacity of the leakage is 2.3 when there are 3 concrete users in the hashmap, and the username contains 3 characters. SPF doesn't scale to larger sizes of the secret, since the hash function causes a path explosion problem.” Vanderbilt concluded that with the budget of 100000, this timing channel can reveal the username.

Evaluation: Vanderbilt identified 3 SC Time vulnerabilities within the challenge program. The first vulnerability was in the isEqual method; Vanderbilt correctly concluded that this side channel was not strong enough to satisfy the operational budget.

The second vulnerability in the Add and Search methods of the GeoMultiTrie class can be used to identify the location of a SubSpace user. Vanderbilt stated that the location of the user was sufficient to impersonate said user. The side channel discovered by Vanderbilt can leaks a user’s location if the user is stationary and the SubSpace database does not change over the course of the attack. These constraints are not provided in the challenge question. Secondly an attacker can use a victim user’s location to impersonate said victim user. The SubSpace application allows users to chat with the closest subspace user. An attacker would therefore have to identify the location of the user closest to the victim user. An attacker can use the side channel in the GeoMultiTrie class to identify the closest user to the victim user; however, this would require that the SubSpace database not change during the attack. As stated above, this constraint is not provided in the challenge question.

The third vulnerability identified by Vanderbilt was in the Database class. Vanderbilt correctly observed that knowing a user’s username is sufficient to impersonate the user. Vanderbilt identified a vulnerability in the src.main.java.com.example.subspace.Database.java file class. The *registerUser* method calls the containsKey method for the usernameToUser hashmap and the emailToUsername hashmap. The call to the usernameToUser hashmap is the only one which contains the username of a given user. The registerUser method determines if a username exits or an emailAddress exits. The logic in the registerUser method leaks information about whether or not a submitted username exists. This side channel is not sufficiently strong to leak the username of a user. The method of “lookup” used on a submitted username is the java.util.HashMap.containsKey method. Vulnerabilities within this method which leak additional information are outside of scope for STAC.

The EL does not believe that any of the three SC Time vulnerabilities identified by Vanderbilt are strong enough to satisfy the operational budget.

Post-Engagement Analysis Ruling: Incorrect

A.2.8.2.13.2 Question 042 (SC Space, Intended Vulnerable, Answered Not Vulnerable)

“The active operation is to issue an HTTP request and receive a response, and the only HTTP request that is related to the location is "update location". After the client issues request to update the location, the server returns the same message "Location will be updated shortly" regardless of

the values of the longitude and latitude to be updated. This means the response size is the same, and thus there is no side channel in space." However, Vanderbilt "figured out an attack that can reveal the information about the location of a user. The attacker registers two (fake) accounts: A and B, and sets the locations of A and B to be a small distance **r**. If the attacker sends hail from account A, and receive the same message from account B, there is no other user within the range **r**. If B does not receive the message, it means there is another user located less than **r** from A. The attacker slowly increases **r** until she finds such a user."

Evaluation: Vanderbilt correctly identified the intended SC Space vulnerability but concluded that this did not qualify as a SC in Space. Since we believe it does qualify as an SC Space vulnerability this response received a post-engagement analysis ruling of correct.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.14 TextCrunchr 1

A.2.8.2.14.1 Question 021 (AC Time, Intended Vulnerable, Answered Yes)

"There is a vulnerability in the ZIP decompressor in the decompress method. This version of TextCrunchr does not check the **total number** of files uncompressed in method decompress, but instead the condition is on the size of a file queue where elements are iteratively added and removed. […]

When an archive is decompressed, all files inside are added to the queue. So, if [one] can keep the queue size between 1 and 15, the loop will keep executing." Vanderbilt "verified that this is indeed a vulnerability by using the SPF Worst Case Analyzer. The approach used was" to "restrict inputs to ZIP files. ZIP files can be nested, and they will be iteratively decompressed by the *ZIPDecompressor*. Whenever a zip is decompressed, all its files are added to abovementioned queue. In some sense storing iteratively the contents of the archives and processing them in this FIFO order makes a breadth first traversal of the file tree. The idea is to generate a zip file with a nesting level controlled by symbolic variables. At maximum, [Vanderbilt] allowed a nesting level determined by the input size variable which is supplied in the SPF driver. Intuitively, the longest path is obtained when the input size bound on the nesting levels is reached. To do this, [Vanderbilt] replaced the ZipInputstream with [their] own "model" that uses symbolic variables for returning a new ZipEntry or null. If null is returned, it means that the bottom of the nesting levels has been reached, and the program will return."

The results showed" that the number of files (or zip nesting levels) to be processed seems unbounded. Since the number of files processed is linear in the maximum nesting level allowed: In other words, an attacker can keep adding new elements to the queue without violating the loop condition this.queue.size() < 15. Since decompressing is a relatively expensive operation, the intuition is therefore that the vulnerability can be exploited by constructing a zip file with sufficiently many nesting levels such that the time bound (500 seconds) is exceeded. The other condition to take into account is the number of bytes read from the nested archive structure. This bound is ~5 MB, but by using a zip quine that extracts an exact copy of itself, one can obtain a very large number of zip decompressions while still being without the bound." Vanderbilt provided a working exploit.

Evaluation: Vanderbilt correctly identified the intended vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.15 TextCrunchr 2

A.2.8.2.15.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt used “the Janalyzer to load the program, which first shows the call graph. Since this question is about complexity in time,” Vanderbilt went through all the “inter-procedural nested loops identified by the Janalyzer to gain some intuition about the program’s behavior.” Vanderbilt “found this variant is different from others in terms of how the hashmap is implemented. For example, in other variants, when the put() method is called, it may call putTreeVal() , but in this variant, it does not use redblack trees to store colliding elements. This raised their suspicion. By checking the decompiled source code, [Vanderbilt] found as long as all the strings can map to the same bucket, the dictionary operations run for O (n) in the worst-case. [Vanderbilt] verified [their] suspicion by generating a file with strings whose hash values all collide and testing it on the reference platform[…] Here method hash simply rehashes the hash from hashCode() (in this case String.hashCode()), but that will of course still yield the same hash for two different objects that have the same hash like "Aa" and "BB" where hashCode() returns 2112. Analysis using the SPF Worst Case Analyzer shows that complexity of lookup in the HashMap is linear with respect to the number of objects in the map, *irrespective* of the initial hashmap size.”

“The SPF Worst Case Analyzer results show that it is possible to **always** obtain O (n) lookup time in the hashmap. What makes this a vulnerability is the fact that the user can **control** the number of elements that are added since it is used in many of the text processors, notably WordFrequencyProcessor where it stores the frequencies of each word in the input file. In practice, the input file can be very large because the input file can be b-zipped or zipped. To exploit the vulnerability, Vanderbilt needed to construct an input file consisting of (preferably all) unique words that hash to the same value according to String.hashcode().

The idea is that if two strings, A and B hash to the same value, then A + B and B + A hash to the same value (here + concatenates). More generally, if a length N is set on the word size, then all permutations of the alphabet {alpha, beta} in words of size N will generate the same hash, but – obviously – be unique. For instance, if the alphabet is {"Aa", "BB"} (again, both hash to 2112), then if N = 3 the following words will generate the same hash:
AaAaAa AaAaBB AaBBAa AaBBBB BBAaAa BBAaBB BBBBAa BBBBBB
The alphabet can be regarded as the base in the conversion. If the alphabet is extended, then the sequence will be even longer for the same N – in some sense, the greater the alphabet is, the more compact an attacker can make the words thus increasing efficiency, i.e. increase the number of words that can be submitted.”

Evaluation: Vanderbilt correctly identified the vulnerability within the application

Post-Engagement Analysis Ruling: Correct

A.2.8.2.16 TextCrunchr 3

A.2.8.2.16.1 Question 011 (AC Time, Intended Vulnerable, Answered Yes)

"As the results for question 008 pointed to the sorting algorithm, Vanderbilt studied the one in this version first. The sorting algorithm in this version appears to call *changingSort()* first, which does not result in a sorted list. Then, *quicksort()* is called.

The SPF WorstCaseAnalyzer showed the order of the entire sorting algorithm (combination of both) is $O(n^2)$. [Vanderbilt] analyzed the QuickSort algorithm in isolation, to see if it contained any particular vulnerabilities. The results revealed that the program behaved as expected. The worst case complexity was $O(n^2)$ and the WorstCaseAnalyzer showed that this was the case for an already ordered list in which all elements occur only once. "

"The fact that the two sorting algorithm are combined makes this program slightly harder to analyze. In order to find the 'global' worst case input, [Vanderbilt] used the output of their first analysis, where the combination of both algorithms was studied. That showed that, contrary to the result for quicksort in isolation, the worst case is when the list of words is in reverse order. To check if this worst case amounts to a vulnerability as defined by the given budgets," Vanderbilt "created a file with 315000 'words' (i.e. 4-letter combinations) in reverse order. When zipped, this file has a size of just under 400kb, just within the budget. Running TextCrunchr 3 on this file takes 6 minutes and 44 seconds on the reference platform."

Evaluation: Vanderbilt correctly identified the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.2.8.2.17 TextCrunchr 5

A.2.8.2.17.1 Question 017 (AC Space, Intended Not Vulnerable, Answered No)

"For this challenge, [Vanderbilt] studied the components for which [they] found vulnerabilities in the other TextCrunchr versions. [Vanderbilt] used the SPF WorstCaseAnalyzer to analyze the sorting algorithm's memory consumption. […] Apart from some strange bumps, the results are polynomial with respect to the length of the list. In order to confirm that this cannot lead to exceeding the budget, [Vanderbilt] ran the program on the exploit file for TextCrunchr 3 (a text file of maximum size, with a list of 4 letter combinations, in reverse order). This consumes < 240 MB of memory, which is within the budget of 512 MB. […]
Next, [Vanderbilt] did the same analysis for the HashMap class. The resulting graph shows memory consumption that seems to match what is expected: linear increase, then a larger increase as structure changes. Another aspect of [their] analysis is that also constraints are output that exercise the worst case path. These show that the worst case is when all hash values are equal. For this reason, [Vanderbilt] ran the program on the input for TextCrunchr 2 (see Question 37). Running TextCrunchr 5 with this input file uses < 200 MB of memory, well within the budget of 512 MB."

Evaluation: Vanderbilt correctly identified that the application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.17.2 Question 024 (AC Time, Intended Not Vulnerable, Answered No)

"[Vanderbilt] first applied the Janalyzer to find deeply nested loops / allocations. Both the loop nesting level and the deeply nested allocations point to the HashMap. This is also where [Vanderbilt] found problems in other versions of TextCrunchr, so they tried running those previously found attacks first. The program stays within resource bounds, even with the large file that causes collisions in the HashMap. […] Since [Vanderbilt] could not find anything in the HashMap, they also had a look at the (previously problematic) sorting algorithm. The graph for Sorter.sort has some strange bumps, but overall seems quadratic, as expected. […] [Vanderbilt] then ran the Fuzzer on the sorting algorithm in isolation, feeding it files of various word count (from 2 to 9999) and word lengths (3, 6, and 20) in sorted, reverse sorted and random orderings to determine if it exhibited a similar vulnerability that several of the other versions did. All test times were < 1 sec, so there did not appear to be any time-related vulnerability in this version of the *Sorter*."

Evaluation: Vanderbilt correctly identified that the application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.8.2.18 TextCrunchr 6

A.2.8.2.18.1 Question 008 (AC Time, Intended Vulnerable, Answered Yes)

Vanderbilt "first applied the Janalyzer, using Averroes and the 0CFA option. This resulted in a list of inter-procedural nested loops and "new" memory allocations Most of the loops reported by the Janalyzer are related to the HashMap class with the put and treeify methods being very common in the branches. However, there were three loops and two "new" allocations in the 4-level that indicated the Sorter that included the *doIt1* method of the ClasschangingSort and Classmerge classes were involved. It also indicates that, for the three loops, the loop exit condition changes within the loop. Analysis of the HashMap revealed no problems, so [Vanderbilt] studied the sorting algorithm instead. Running the SPF WorstCaseAnalyzer on the sorting algorithm shows strange behavior when the length of the list is a factor of 3. Running the SPF WorstCaseAnalyzer with values 3, 6, 9… 24 shows that the algorithm's behavior is approximately exponential for those values. This indicates that there is a vulnerability related to sorting lists of size N, where N is a multiple of 3. The sorting algorithm is called from the WordCountProcessor to sort occurrences. It can be triggered by using a file with N different words in reverse sorting order where N is a multiple of 3. For instance, for a single text file consisting of the words t1 to t6000 in reverse order, the program runs for over 5 minutes, yet when t6001 is added, it terminates within a second. The ordering does not have to be in reverse order. A file of 9999 entries that are in increasing or decreasing order both have runtimes much greater than 5 minutes. It seems to be due to the oddity in the merge sort that is used. When doing the split in Sorter.ClasschangingSort.doIt1, the loop checks if the list size is an even modulo of 3, and inserts an extra split in each loop."

Evaluation: Vanderbilt correctly identified the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.2.8.2.19 TextCrunchr 7

A.2.8.2.19.1 Question 033 (AC Time, Intended Vulnerable, Answered Yes)

"[Vanderbilt] used [their] Janalyzer tool to load the program, which first shows the call graph. Since this question is about complexity in time, [Vanderbilt] quickly went through all the inter-procedural nested loops identified by Janalyzer to gain some intuition.
This program is like other variants, and our tool again flags the most suspicious parts is being in the hashmap implementation. By manually inspecting the decompiled source code, Vanderbilt found the implemented hashmap uses chaining to resolve collisions at the beginning, and when TREEIFY_THRESHOLD (8) is passed, treeify() establishes a balanced red-black tree. After this, newly added elements that collide will be put into the red-black tree.
From the call graph," Vanderbilt observed "that when a new element is inserted into the tree through putTreeVal(), it does not call the balanceInsertion() method which calls rotateLeft() / rotateRight() to necessarily maintain red-black tree properties, so it means the tree can be degenerate (that is, dictionary operations can run O(n) in the worst case)." Vanderbilt believed that this was the location of the vulnerability.

Vanderbilt verified their "suspicion by generating a file with strings whose hash values all collide and testing it on the reference platform. Analysis using the SPF WorstCaseAnalyzer shows that complexity of lookup in the HashMap is linear with respect to the number of objects in the map, irrespective of the initial hashmap size. The SPF WorstCaseAnalyzer results show that it is possible to always obtain O (n) lookup time in the hashmap." Vanderbilt added that "what makes this a vulnerability is the fact that [they] can control the number of elements that are added since it is used in many of the text processors, notably WordFrequencyProcessor where it stores the frequencies of each word in the input file. In practice, the input file can be very large because the input file can be b-zipped or zipped."

To exploit the vulnerability, Vanderbilt constructed an input file consisting of (preferably all) unique words that hash to the same value according to String.hashcode(). Vanderbilt generated and provided a working exploit.

Evaluation: Vanderbilt correctly identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9 Invincea (Control Team)

A.2.9.1 Invincea Overview

Invincea answered all of the Engagement 2 questions with a 74% accuracy. Invincea had an 86% accuracy and a 70% accuracy for SC Space and SC Time vulnerabilities respectively.

Table A-40: Engagement 2 Invincea Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	7	6	86	6	86
SC Time	10	7	70	7	70
SC Space/ Time	1	0	0	0	0
AC in Space	8	6	75	6	75
AC in Time	16	12	75	12	75
Total	**42**	**31**	**74**	**31**	**74**

Table A-41: Engagement 2 Invincea Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	23	15	**65**	15	**65**
Not Vulnerable	19	16	**84**	16	**84**
Yes Answer	18	15	**83**	15	**83**
No Answer	24	16	**67**	16	**67**

A.2.9.2 Invincea Specific Responses

A.2.9.2.1 Blogger

A.2.9.2.1.1 Question 016 (AC Space, Intended Not Vulnerable, Answered No)

Invincea responded that "this question is the same as question 40 except it is space-based, rather than time-based. Question 40 had a very simple and obvious solution. However, after significant effort looking through the reverse-engineered java code, no suitable POI for a space-based solution was identified. Therefore, this challenge binary is not vulnerable to a space-based algorithmic complexity vulnerability."

Evaluation: Invincea correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.1.2 Question 040 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that "the approach to finding POIs was to run the example-input.sh file and look for CPU hotspots via JProfiler's CPU Hotspot window. After noting the URIVerifier.verify hotspot, a long input to the verify method was attempted based on intuition. A vulnerability was found in URIVerify.verify(). A long UIR (sic) of the same value will easily trigger the vulnerability."

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.2 GabFeed 1

A.2.9.2.2.1 Question 007 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that" there is a time-based algorithmic complexity vulnerability found in the breakLines member of the LineBreak class in the message renderer. It is triggered by submitting a large collection of very small tokens which forces the string splitter to iterate over the entirety of input numerous times in nested loops. By providing the input ('A '*140000), [they] exceeded the required runtime of 5 minutes, while keeping total request PDU size under 40000 bytes."

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.2.2 Question 010 (SC Time, Intended Vulnerable, Answered No)

Invincea responded "that Question 10 requires an attacker to discover the server's private key used in authentication via a side-channel in space. Available actions are observation of encrypted packets, issuing an http request and receiving a response, and querying an oracle. Each of these actions costs a point, with a total operational budget of 60,000 points and a required 95% probability of success for an attack approach to be considered valid.

There is no side-channel permitting discovery of the server's private key. The private key is used in only one part of GabFeed: during the initial authentication, a user supplies their public key and GabFeed responds with a master secret, generated via an algorithm that repeatedly uses multiplication and a hard coded modulus of the user's public key. The master key is not used in any subsequent interactions between the user and GabFeed, severely limiting the attack surface.

Observation of the encrypted packets yields no significant information about the server's private key--only the user's public key and the generated master secret is transmitted, so only rough inferences regarding the size of the two values can be made.

Issuing an HTTP request and receiving a response similarly yields very little information--an attacker can probe the range for the master secret, but as the hard coded value used for the modulus operation is 1536 bits long, the available range for the master secret is too large for brute force attacks within the operational budget. The algorithm for generating a shared secret from the master key and the client key is not sufficiently weak to expose more than a very slight amount of information about the private key; an attacker may be able to infer if a server has a key with no set bits, but extracting any additional information is impossible.

Without additional information, querying the oracle is fruitless, as the master key may be arbitrarily large and brute force guessing the secret is not tenable nor uses any side channel information.

The secret key is not used in any other part of the server's operation, is never exposed outside of the private operation to generate the master secret, and the master secret's only use is to be presented to the user before login. Given all of this, there is no side channel attack in time that will expose the server's private authentication key."

Evaluation: Invincea did not identify the intended SC Time vulnerability in the application. In unison the modPow method in com.cyberpointllc.stac.math.ModPow and the fastMultiply method leak the server's private key.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.2.3 Question 029 (SC Time, Intended Not Vulnerable, Answered No)

Invincea responded that "the challenge for question 29 is to learn a secret about a user via observation of encrypted packets, http requests to a server, and confirmation via oracle. Each operation costs 1 point, and the max budget is 100,000 points and a 95% success rate is required for a successful solution. In this case the username is already known, so retrieval of the user's password is sufficient.

This is a very narrow scope of interactions available. Notably, the user supplies their username and password, which is checked by the server before creating a session for the user. The password check algorithm encrypts the supplied password with the server's master key, then compares the encrypted value with the stored encrypted value. The length of the two encrypted values is checked, then the characters of the smaller of the two encrypted values are compared one at a time against the characters of the other encrypted value. If either the length or any character does not match, the returned value is false.

There is a timing attack that will disclose the length of the correct password: as the algorithm will only compare as many characters are in the shorter of the guessed and stored passwords, the number of comparisons has an upper bound of the length of the correct password. Thus an attacker can make login attempts with increasingly longer passwords, timing the delay in response, until lengthening the password no longer increases the delay, as the supplied password is now longer than the stored password. A scripted attack of this nature is provided though its accuracy as it is attempting to detect the time taken by a single extra character comparison is highly dependent on collecting a great deal of data.

There does not appear to be a timing attack that will disclose the actual contents of the password, as the password check algorithm will check the length of the passwords and each individual character regardless of correctness of the password or the first mismatch between the guessed and stored password."

Evaluation: Invincea correctly responded that the challenge application did not contain an intended SC Time vulnerability. Invincea did report a vulnerability which leaks the length of a user's password.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.2.4 Question 038 (AC Space, Intended Not Vulnerable, Answered No)

Invincea responded that" the challenge for question 38 is to cause the application to consume more than 512 MB of memory via input of no more than 400,000 bytes. There are a couple of potential weak spots in the code, the most obvious of which is the LineBreak class. However, it will only allocate 16 bytes per token in the input it receives, and so would require approximately

30 million tokens, far exceeding the input budget. The next candidate is the TreeNode class, but while the implementation is complex, it is safely written with respect to memory usage."

Evaluation: Invincea correctly responded that the challenge application did not contain an intended AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.3 GabFeed 2

A.2.9.2.3.1 Question 006 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that the GabFeed Hash Function was vulnerable.

"public static int hash(Object key, int capacity)
 {
  return (key.hashCode() & 0x7FFFFFFF) % capacity;
 }
Where 'capacity' is the static size of the table and 'key' is the input value supplied."

"The GabFeed hash implementation depends on a few factors: the value itself, the existing size of the table, and the capacity. Invincea observed that exploiting a vulnerability required a large collection of values and loaded the GabFeed hashing implementation in a Jython instance and exercised in a brute force fashion to find colliding values. The file was then written to disk and sent to the application server via curl. A Jython script was used to generate colliding hash value inputs."

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.3.2 Question 032 (SC Time, Intended Not Vulnerable, Answered No)

Invincea responded that the goal was to leak the server's private key. "At first glance it seemed that the master secret could be recovered from timing differences caused by the branching in the modPow() and fastMultiply() functions that occur on each bit in the private key." Invincea observed that "due to this particular implementation there does not appear to be a way to leak the server's private key".
Invincea added that "the ModPow class […] uses a version of the Montgomery powering ladder explicitly designed to mitigate side-channel timing analysis. This is due to the near constant timing of both branches of the if-statement, this regularity makes measurements of these branches negligible. The current state of the art for timing analysis for this sort of vulnerability is based around the idea of carefully choosing base values that will leak data about the private key. Such an attack is mitigated here by taking the modulus at every step in the if-block and multiplying r1 and r0 in both stages, thus making values to do this nearly impossible to find."

Evaluation: Invincea correctly concluded that the challenge application did not contain an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.4 GabFeed 3

A.2.9.2.4.1 Question 023 (SC Time, Intended Vulnerable, Answered Yes)

Invincea responded that "the side channel in time is […] in the handlePost method of the SearchHandler.class: if (this.dailyTerms.contains(searchTerm)) {…}. […] There is a clear branch in execution depending on if the target's search term was a secret word or not. It takes more operations if the search term is a secret word."
Invincea added that "since an attacker has unlimited operations to interact with the server, the attacker can manipulate the server to a state where all search terms will have unique timing signatures. The attacker can then sniff encrypted packets and match packet timings with his/her collection of timing signatures to identify when a user has searched for a secret term."
Invincea provided a working exploit script and evidence of the observed timing differences.

Evaluation: Invincea correctly identified the intended SC Time vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.4.2 Question 036 (AC Time, Intended Vulnerable, Answered No)

Invincea responded that using "CPU profiling several methods in the com.cyberpointllc.stac.linebreak class were identified as taking exceptional amounts of processing time. In particular the interaction between breakLines(), search(), searchHelper(), and cost() methods composed the only obvious point of interest. Invincea had found an algorithmic complexity vulnerability in GabFeed 1 they attempted the same exploit. The exploit was to upload a large message to a conversation thread that was thousands of single letter words. After testing many different large messages using different numbers and lengths of words, it was concluded that this version of the LineBreak class exhibited the same style of behavior, greater number of words equaled greater runtime, but with a much less dramatic effect.
Invincea concluded that there was not a way to generate the vulnerable effect within the input budget."
Invincea looked for other points of interest without success and determined that the application was not vulnerable.

Evaluation: Invincea did not identify the intended AC vulnerability in the challenge application. The sorting algorithm used in this variant of GabFeed is vulnerable. The sorting method's slow mode is triggered when the number of items to be sorted is divisible by 8.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.4.3 Question 039 (SC Time and Space, Intended Vulnerable, Answered No)

Invincea responded that "when searching for a special term, the user receives a blurb describing the term. An attacker can permute all special terms to receive their blurbs. The blurbs are all different lengths and are prepended to all search results.

Lengths from term file: malaria: 636; HIV: 790; dyslexia: 858; lupus: 874; Multiple Sclerosis: 908;

fibromyalgia: 1052; anemia: 1478; pericarditis: 1925; gonorrhea: 2547; ringworm: 2763; dermatophytosis: 2781; polio: 2914; poliomyelitis: 2914; diarrhea: 3139; angina: 3179; epilepsy: 4059; cancer: 4138; diabetes: 4701.

An attacker has access to the terms, and searching for terms in succession will let her collect all description lengths, and the length of search results in case there are message or thread results that also match. For easier analysis, the attacker can record her own traffic to correlate term description size to encrypted packet size.

After this is collected, the attack becomes simple: watch for encrypted traffic and match the size of the results to the recorded lengths. Some terms are close enough in size (such as dyslexia and lupus) where the attack might not succeed, but there are enough large deltas to achieve 90% success rate.

Measurement of http packet indicates if more or less information is being passed, leaking the fact that it is a special term or not."

Evaluation: Invincea did not identify the intended Space/Time SC vulnerability in the application. In Question 23 Invincea correctly identified the independent SC Time vulnerability which allows an adversary to determine whether a user operation is a search for a special term or another GabFeed operation. Invincea's response indicates that they identified the independent SC Space vulnerability which allows an adversary to distinguish individual special search terms. Invincea stated in their response that this SC Space vulnerability satisfied the operational budget but responded that the challenge application was not vulnerable. Invincea responded that the challenge application was not vulnerable. Invincea did not identify the paired SC Time/SC Space vulnerability. Invincea did not identify the SC Space vulnerability which depends on mapping the size of the output packets to special search terms and using the size of the input packets to remove collisions with the user adding to the database.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.5 GabFeed 4

A.2.9.2.5.1 Question 003 (AC Time, Intended Vulnerable, Answered No)

Invincea responded that "POIs for algorithmic complexity were in the sorter class. In this application the two functions which stood out were the helper functions (changingSortHelper and mergeHelper), as they were both custom and dealt directly with the core functionality of sort() and the optimizations associated with it." Invincea concluded that "both POIs did not appear to be exploitable in any way, nor did their edge cases appear to be useful."

"In TreeNode/Hashmap.class, rotate, treeify the hashmap, and all balancing functions" were POIs for complexity. Invincea stated that "just like the sort function, hashmap custom functionality is a good target for AC attacks. However in this case, the normal worst case time associated with hashmaps (i.e. flatten to a list traversal on lookup) was not trigger-able given the addition of the Tree structure, because this is a known mitigation to this attack."

“The SearchHandler functionality and Chat thread functionality […] seemed promising, as search can have a worst case of visiting every element, and chat could also have a worst case of visiting every message or something similar when a chat log is queried. However these ideas did not appear to be possible in this version of GabFeed.”

Invincea attempted “to exploit the lack of a check for multiple connections by spawning many request threads without waiting for a response and after the last one is sent, listen for a response. Initially the responses will be quick to return, but after n requests sent they start to slow until a delta is reached where after n requests, the replies take over the threshold time to return. This method could still probably work based on the structure of the code, but is not fully scriptable using the current python threading methods.”

Invincea concluded that the challenge application did not contain an AC Time vulnerability

Evaluation: Invincea did not identify the AC Time vulnerability in the hashmap implementation. The hash table is based on a red black tree which is not balanced on ‘put’ calls.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.6 GabFeed 5

A.2.9.2.6.1 Question 001 (SC Space, Intended Not Vulnerable, Answered No)

Invincea responded that “the timing and content of the encrypted packets yielded no data regarding the actual content of the interactions. Users have a number of interactions with the server available to them, the application’s encryption confounds what ciphertext correlates to which part of a returned page and makes reconstruction of a user's behavior or the data they receive impossible.

Issuing an HTTP request and receiving a response provides no information to a third party attacker. Without a valid set of credentials, the server will redirect all traffic to the login page without performing any other processing and without sending any additional data.

Without additional information, querying the oracle is not helpful, as the range for acceptable usernames is arbitrarily large and brute force guessing the term is not possible in two operations.

Given all of this, there is no side channel vulnerability in space that will reveal the username of a user in a private chat to a third party attacker.

Although not a side channel, if an attacker has valid login credentials, they may simply request the contents of all private chats, as the chat pages have predictable addresses and are readable to all logged in users, regardless of whether that user is involved in the chat. In this case the solution is trivial--the attacker logs in to the system, requests the page for the chat in question, and the usernames are available on the page.”

Evaluation: Invincea correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.6.2 Question 004 (SC Space, Intended Not Vulnerable, Answered No)

Invincea responded that “Question 4 requires an attacker to discover the server's private key used in authentication via a side-channel in space. Available actions are observation of encrypted packets, issuing an http request and receiving a response, and querying an oracle. Each of these actions costs a point, with a total operational budget of 60,000 points and a required 95% probability of success for an attack approach to be considered valid.

There is no side-channel permitting discovery of the server's private key. The private key is used in only one part of GabFeed: during the initial authentication, a user supplies their public key and GabFeed responds with a master secret, generated by taking a hard coded modulus of the user's public key raised to the power of the server's master key. The master key is not used in any subsequent interactions between the user and GabFeed, severely limiting the attack surface.

Observation of the encrypted packets yields no significant information about the server's private key. Only the user's public key and the generated master secret is transmitted, so only rough inferences regarding the size of the two values can be made.

Issuing an HTTP request and receiving a response similarly yields very little information. An attacker can probe the range for the master secret, but as the hard coded value used for the modulus operation is 1536 bits long, the available range for the master secret is too large for brute force attacks within the operational budget.

Without additional information, querying the oracle is fruitless, as the master key may be arbitrarily large and brute force guessing the secret is not tenable nor uses any side-channel information.

The secret key is not used in any other part of the server's operation, is never exposed outside of the private operation to generate the master secret, and the master secret's only use is to be presented to the user before login. Given all of this, there is no space-based side-channel attack that will expose the server's private authentication key.”

Evaluation: Invincea correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.6.3 Question 009 (SC Time, Intended Not Vulnerable, Answered Yes)

Invincea responded that “there is no cost associated with performing an HTTP request and receiving a response. An attacker with valid login credentials can login, collect all of the public pages, and build a histogram of all extant terms and their number of occurrences. The attacker can then interact with the server to manipulate the server state and shift the distribution of terms such that no two terms have the same number of occurrences. This is performed to setup for a fingerprint based attack. The attacker fingerprints the state of the system by collecting the timing for each term in the system.

When a victim user interacts with the system to perform a search, the timing of their request can be compared against the existing fingerprint database. The attacker can then match a search term to the timing fingerprint."

Evaluation: Invincea's response noted that because there is no cost associated with performing an HTTP request and receiving a response an attacker can build database of terms in the public database. A note on this statement is that building a database does not count against the user's budget unless otherwise stated. The active operations only apply to operations actively used in an attack therefore regardless of the operational budget-provided active operations building a database for an attack from public information does not count against the attacker's budget. Invincea's response implies that all user interactions for the purpose of the attack are search operations and that a user will only search for terms in the GabFeed database. Given that these restrictions are not provided and the side channel identified by Invincea does not address these cases, the side channel does not account for the worst-case secret as required by the operational definition.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.6.4 Question 014 (AC Time, Intended Not Vulnerable, Answered No)

Invincea responded the challenge question "requires an attacker to exploit an algorithmic complexity vulnerability in time to cause the server to take more than 600 seconds to respond. The attacker has a maximum resource budget of 400000 bytes of PDU in HTTP requests sent from the user to the server.

There are two candidate algorithms in the target application, a sorting algorithm and a search handling algorithm, but none are implemented in a vulnerable manner. The search algorithm presents a possible attack vector, but the runtime complexity of the algorithm is too small to cause excessive processing within the constraints of the input budget. There are no other significantly complicated algorithms in the target application.

Invincea attempted to 'salt' a search term across many separate messages in an effort to overload the sorting method. Sending as many possible copies of a search term to as many messages as possible to overload the search handling method. Putting as many special keywords in as many messages as possible to overload the search handler." This method did not "yield a significant amount of processing time." Invincea therefore concluded that the application did not contain an AC Time vulnerability.

Evaluation: Invincea correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.7 Graph Analyzer

A.2.9.2.7.1 Question 013 (AC Space, Intended Not Vulnerable, Answered No)

Invincea responded that "the challenge for question 13 is to cause graph analyzer to consume more than 512 MB of memory via an input file of no more than 5000 bytes." Invincea investigated the *graph.dot.DotParser*, *org.tigris.gef.base.CmdSavePNG*, and

org.tigris.gef.base.CmdSavePS classes. Invincea provided descriptions of each of these classes and potential memory usage vulnerabilities within each class. Invincea attempted multiple approaches including providing a single node recursive graph, a complete graph, a graph which cycles, and a non-branching cycle of graphs. Invincea concluded that the challenge was not vulnerable.

Evaluation: Invincea correctly concluded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.7.2 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that "there is an algorithmic complexity vulnerability in time in the CmdSavePNG class. The vulnerable code is found in the saveGraphicsLayer member." By giving the program an input that refers to itself it searches by name for the container Invincea caused the application to infinitely loop and recurse over the specified container.

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.7.3 Question 034 (AC Space, Intended Vulnerable, Answered Yes)

Invincea responded that" there is an algorithmic complexity in space vulnerability in the process of outputting postscript files. The primary code of interest is centered on the CmdSavePS class. Invincea exploited the recursive nature of the paintContents code to cause a huge increase in the file output size. Invincea's exploit defines several different graphs with nodes of a container type that corresponds to a lower level graph." By stacking these graphs Invincea created recursive containers in containers that caused the graph output to explode in size.

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.8 Image Processor

A.2.9.2.8.1 Question 005 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that the challenge application uses a doubly nested for-loop to iterate over each pixel in a provided image. Invincea provided an image with 5214 pixels by 3240 pixels and exceeded the resource usage limit.

Evaluation: The challenge application restricts input images to 250,000 pixels (com.stac.image.ImageProcessing class). The input provided by Invincea violates that restriction. The application uses the java.awt.image.BufferedImage class to process all input images. This class takes longer to process images with more pixels. This class is called for all input images. The output object is used to check the application's input guard. If the application recognizes that the input image exceeds 250,000 pixels an exception is thrown. According to the Operational

Definition document for STAC, the branch for ‘slowness’ in complexity vulnerabilities must occur inside the challenge program. In this case the branch for ‘slowness’ occurs in the java.awt.image.BufferedImage class which is an external library. The vulnerability identified by Invincea is not valid by this definition. The intended vulnerability for the challenge program is within the Mathematics.intensify function. The filters.Intensify.filter function iterates through the pixels of a provided image and calls the Mathematics.intensify function for each pixel. The Mathematics.intensify function contains a loop in its *exp* function. The termination point of this loop depends on the value of the pixel provided to the Mathematics.intensify function. An input image of size 500 pixels by 500 pixels containing the rgb value which maximizes the number of loops performed in the *exp* is sufficient to exceed the resource usage limit. Invincea identified an out of scope vulnerability and provided a working exploit

Post-Engagement Analysis Ruling: Correct

A.2.9.2.8.2 Question 030 (AC Space, Intended Not Vulnerable, Answered No)

Invincea responded that “the main focus of this program was the call the program makes to the Clustering function where it 'clusters' the training set of images. The process for this involves two data structures: HashMap and LinkedList.” Invincea observed that the number of images (training and classification) correlate to the memory usage. Invincea attempted several exploits but only managed to use 197.7 MB of RSS memory. Invincea concluded that the application was not vulnerable.

Evaluation: Invincea correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.9 Law Enforcement Database

A.2.9.2.9.1 Question 019 (AC Time, Intended Not Vulnerable, Answered No)

Invincea responded that “this question requires an attacker to discover an algorithmic complexity vulnerability that will cause the server to run for more than 30 seconds before completing a response to the attacker. The attacker has a resource budget of 1000 bytes of UDP payload data.

There are a few notable classes where such complexity could exist, most notably the BTree class. Inspection of that class, however, shows no vulnerability, and no inputs constructed cause pathological behavior. Additional potential locations exist in the GET and PUT cases of the packet handling code in the UDPServerHandler class, but they likewise seem to be safely implemented. Other candidate locations also bear up under scrutiny, and running in a debugger yields no hotspots for processing. Given the relatively small input budget, no vulnerability exists that will cause the server to process for the required 30 seconds.

When testing in a virtual machine, a quirk of the server and the virtual machine loopback interface can cause processing to take more than 30 seconds for simple searches, but when running on the target platform those same searches complete in less than 1 second.”

Invincea concluded that the application was not vulnerable.

Evaluation: Invincea correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.9.2 Question 026 (SC Time, Intended Vulnerable, Answered Yes)

Invincea responded that "the client can perform a SEARCH operation. The client provides a range of IDs to search to the server. The server retrieves all IDs in the range, performs various operations, and then returns all UNRESTRICTED IDs within the range to the client. The server performs operations on DFileHandle log files. These operations vary depending on the SEARCH operation results. [An attacker] can also detect these operations as they are sent over the network on PORT 6666." Invincea noted that while iterating through the list of IDs in a search range UNRESTRICTED IDs are immediately returned while RESTRICTED IDs are not sent but an exception is thrown. Invincea provided an exploit which uses this vulnerability to identify the RESTRICTED ID in the provided database.

Evaluation: Invincea correctly identified the intended vulnerability within the application.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.9.3 Question 027 (SC Space, Intended Not Vulnerable, Answered Yes)

Invincea's response was similar to the response for Question 26. Invincea added that in the case of an unrestricted ID the server writes to the "lastaccessinfo.log" and in the case of a restricted id the server writes to the "lastacces.log" file. Invincea's exploit is to run a binary search by detecting the additional DFileHandle("lastaccess.log") operation.

Evaluation: The response by Invincea is incorrect. The operational budget does not provide any passive operations for an attack. Observing the additional DFileHandle("lastaccess.log") operation requires either observing packets sent on port 6666, observing the "lastaccess.log" file, or observing the memory usage of the application. All of these operations are passive. In addition the description document places files in the server directory out of scope. The "lastaccess.log" file is in the server directory and is therefore out of scope. The application did not include an intended SC Space vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.10 SnapBuddy 1

A.2.9.2.10.1 Question 012 (SC Time, Intended Not Vulnerable, Answered No)

Invincea responded that "after thorough analysis of decompiled bytecode and dynamic analysis, it was determined that SnapBuddy 1 does not leak sufficient information to determine a user's friends.

Very little information can be gleaned from a user's encrypted communications with the server that would be useful in determining another user's friends. Any data provided by the server to an authenticated user is user-specific, and controlled by the session associated with a user's session

cookie. It is not possible as a logged-in user to see another user's friends. It is however possible to see the image assets that will load for all users on the add friends list. Because these images are rendered by the server for each request of the image, the server does in fact leak information identifying some users in the render time of their thumbnail.

On its own, the total round-trip time on an HTTP request is not that interesting. Even if one assumes the ability to identify image requests by their place order of all requests to load a page, or the time that they take to process, without some way to differentiate those photos' download times, the timings themselves are meaningless. The server does expose to an authenticated user the ability to observe all non-friends' image thumbnails – the same thumbnails that would be loaded on the friends view. By observing the timing of all users' profile thumbnail download times as reconnaissance, an attacker would be able to build a rough lookup table of image response times correlated to a user. It should be reiterated, here, that this attack vector is only feasible due to the server's rendering of each image's filters for each request of that image. The lack of caching exposes the potential side channel.

Armed with a listing of every user's profile thumbnail download time, an attacker might be able to note the download times of a user's friends and lookup the times on their precomputed reference. Unfortunately, this attack proves infeasible because there is not enough variance in the interestingly long-running image download times to single out a user."

Evaluation: Invincea correctly concluded that the challenge application did not contain an intended vulnerability. Invincea identified a side channel but ruled that it was not sufficiently strong to satisfy the operational budget.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.10.2 Question 025 (AC Space, Intended Vulnerable, Answered No)

Invincea responded that "the sort and hashmap POIs are the same as the other SnapBuddy binaries, and do not reveal an AC vulnerability into the program. The only difference was in Sort with no default comparator class override. Invincea added that the image services were of interest as they by default forced the highest memory usages by an order of magnitude of all the handlers. Invincea attempted to take advantage of addphoto implementation. The program only performs a size check of the new photo object after it is created, this means that by passing an image greater than the size check you can still cause allocations in memory. However as it is unclear if the image itself is sent in the PDU of the HTTP request, and the size check is exactly the constraint of the question, Invincea counted this as a non-solution and just a bug in the code."

Evaluation: Invincea did not identify the vulnerability in the add filter methods of the application. Each filter operation uses memory; in order to prevent users from exceeding the resource usage limit, there are a limited number of filters which users can use and users cannot apply duplicate filters. The duplicate filter check for the 'ScaleUp' filter is case sensitive, an attacker can specify the same filter multiple time by using a different combination of cases for each letter in the command. Using this method the attacker can exceed the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.10.3 Question 031 (SC Space, Intended Vulnerable, Answered Yes)

Invincea observed that the user's location is set with BSSIDs and the POST request to set a location contains the BSSIDs. Invincea also noted that the sizes of the BSSID sets for various locations vary and that it is possible to map the size of the BSSID set to a given location. Invincea provided an exploit which uses this information to leak a user's location.

Evaluation: Invincea identified the SC Space vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.11 SnapBuddy 2

A.2.9.2.11.1 Question 002 (SC Space, Intended Not Vulnerable, Answered No)

Invincea responded that "a user is authenticated by the server by the 128-bit UUID session cookie. This is established when the user provides valid credentials through the login handler. The user's credentials are sent to the server contained in a POST request. The server then issues a valid/invalid response.

It is possible to leak the size of a given user's password. (If [the attacker] know[s] the user is interacting with the system & their username). This would involve fingerprinting the login POST request and the server's valid response. An attacker could then monitor encrypted packets, identify when the server sends the valid response, and then retrieve the previous packets consisting of the POST request. From this the attacker can calculate the size of the user's password that was used to trigger the valid server response.

[Invincea added that] there does not appear to be a space side channel attack that can leak the actual contents of the password.

[Invincea noted that] there also does not appear to be a space side channel attack that can leak an authenticated user's UUID session cookie. The cookie size is constant and the server response is a redirect for an invalid session when trying to access authenticated pages.
Invincea concluded that the application is not vulnerable."

Evaluation: Invincea correctly responded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.11.2 Question 015 (SC Time, Intended Vulnerable, Answered No)

Invincea responded that "the user supplies his / her public key via a GET request to the login page. The server then takes this value and performs com.cyberpointllc.stac.auth.KeyExchangeServer.generateMasterSecret(BigInteger clientPublic) which calls com.cyberpointllc.stac.math.ModPow.modPow(clientPublic, secretKey, modulus). The result of this computation is returned to the user. There is also a timestamp that shows the server's HTTP exchange processing time. The modPow & fastMutiply methods jumped out as POIs because they were custom implementations of existing java.math.BigInteger functionality. […]

The modPow function implements the square-multiply algorithm. It performs OptimizedMultipler.fastMultiply to multiply when the exponent bit is set.

fastMultiply has 3 cases with varying runtimes. (Ignoring cases where x or y is 1)

Case 1: This is an operation using the built in BigInteger.multiply, it seems to be the fastest branch
Case 2: This operation has a runtime that linearly scales with yLen
Case 3: This operation performs two operations of Case 1.

The square-multiply algorithm has been frequently used as a power analysis side channel example." Invincea concluded that the provided model is a "client server model […] where [an attacker] can only obtain the duration of the HTTP exchange / entire calculation. [… An attacker is] unable to obtain intermediate timing for individual bits.

Option 1:

The first goal is to obtain the # of 1's that are set in the private key. This can be determined by choosing a "y" value where yLen > (modulus.bitLength + 32). This would ensure that Case 2 is always triggered when fastMultiply is called. The attacker can then obtain the timing of a single Case 2 operation for our "y" input through instrumenting on the duplicated server. Using the "y" input on the original server, the attacker should get a timing result of "N" * (time for a single Case 2 operation), where N would be the # of 1's in the private key. It is actually best to choose an exorbitant value for "y" to amplify the Case 2 operation time and minimize noise. yLen = 15000 worked well in test cases.

Now that [the attacker knows] the # of 1's in the private key, [the attacker] can start targeting individual bits. [The attacker needs] to find an input for each bit position K that triggers Case 1 only for K, and proceed to trigger Case 2 for all bits > K. [The attacker] can then use the same technique as before, albeit with more noise, to obtain a timing for the # of 1's in the private key. However this time, the bit in position K has been forced to run fast no matter if it was set or not. So if [the attacker gets] an N value of N-1, it means that bit K is a 1 and the effect of bit K on the overall timing has been negated. If [the attacker gets] N = N, then bit K is a 0 which had no effect.

Option 2:

In the first option, [Invincea] looked for an input that would make bit K fast, with bits > K slow. This requires knowing the # of 1s since the attacker has to negate the effect of a bit to identify its value. Another option is to make bit K slow, and bits > K fast. Since the attacker already know the timing of bits (0 … K), if the input on bit K increases the timing significantly then bit K is a 1. This option also appears to require inputs > 1536 bits so the noise would also be lower than option 1. There is also no prerequisite knowledge needed for the # of 1s in the private key.

[Invincea] managed to obtain values that satisfy Option 1 for bits 0...1, and Option 2 for bits 0...10 in the best case of bits (1...8) = 0. The value used for Option 2 is the value (modp1536 + 2). If there are intermediate 1s in the private key, then the left over modulus value grows faster

and the attacker won't be able to get as many bit positions. In the example where the leftover value is 2, at iteration 11 it would be 2^2048 which would exceed the modulus and has a remainder of bit length 1536. If there is a value L where L^2048 mod M < 1536 bits, then that would work for a scenario with bits (1...10) = 0. Intermediate 1s in the private key would make the necessary equation much more difficult.

[Invincea was not] able to find a method to leak the full private key. This is due to their inability to find inputs that can be used for the above methods. Invincea concluded that they are unsure if there even exists a set of valid inputs for all values of K."

Evaluation: Invincea appears to have identified the intended SC Time vulnerability but was unable to craft an exploit and concluded that the application was not vulnerable. The challenge application contained an SC Time vulnerability due to two sets of conditionals in the application. In the ModPow.modPow() method there is an if statement where the conditional is dependent on the value of each individual bit of the server's private key. Within the if statement there is a call to the OptimizedMultiplier.fastMultiply() method. This method contains an "if, else if, else" block where the conditional is dependent on the values of the variable "s", and the variable "base" which is the value submitted by an attacker. The side channel which leaks the server's private key is dependent on the differential processing times within the ModPow.modPow method and the OptimizedMultiplier.fastMultiply() method.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.11.3 Question 028 (SC Space, Intended Vulnerable, Answered Yes)

Invincea responded that "the successful login flow is [authenticate] -> [login] -> [set location] -> [confirm location] -> [redirect to friends page]. After the login process, the user is redirected to the "friends" page and is able to interact with the SnapBuddy system.
There is a header and sidebar that persist while the user can navigate to various pages. The header contains the profile picture of the user. The "friends" page contains the profile pictures of the user's friends. When the user interacts with SnapBuddy using a web browser, the browser will automatically perform requests to acquire the image resource. The server will return a response that contains the image resource.

Since profile pictures are public, an attacker can fingerprint the server response for every profile picture. It is also useful to fingerprint part of the login process in order identify when the user is redirected to the friends page. When the user hits the "friends" page, the attacker can then compare the size of the encrypted packets to identify which images / friends were requested.

The user's own profile picture is also requested due to the header on the "friends" page. If the user is unknown, the attacker can continue to monitor the user's session. Since the header is loaded for every page, the most requested image during the user's session will most likely to provide the user's identity."

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.11.4 Question 035 (AC Space, Intended Not Vulnerable, Answered No)

Invincea investigated several POIs including adding photos, adding locations, adding image filters, sending invitations, editing usernames, adding a profile photo, and changing location. Invincea noted that adding image filters they were able to increase the server's memory usage by ~100 MB but not exceed the resource usage limit. Invincea concluded that the challenge application was not vulnerable.

Evaluation: Invincea correctly concluded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.12 SnapBuddy 3

A.2.9.2.12.1 Question 018 (SC Time, Intended Not Vulnerable, Answered No)

Invincea responded that "impersonating a user would require an attacker to issue requests to the server in such a way to convince the server that they are operating as a bonafide user. A bonafide user is identified by the server through the use of a cookie with a user-session token. Each of the server's request handlers processes requests to different paths or functionalities of the SnapBuddy web application. All requests to the server are handled by one of three classes of request handler.

The first class of handlers is for authenticated actions. Before executing the request handler for one of these functions, the server first checks the user-session token against the internal structure of known user-session tokens. Unsuccessful verification of the token is due either to the non-presence of the user-session token in the internal user-session tokens structure, timeout of the session behind the token, or the lack of a cookie (and therefore lack of presented user-session token). It triggers a redirect response (to the unauthenticated login actions) from the server.

The second class of handlers is for unauthenticated actions. Used to direct a user through a sign-in flow, the server does not check for a valid user-session token when processing requests using these handlers. Upon successful login using these handlers, the server sends the user-agent a cookie containing a user-session token and stores the token in an internal data structure for later verification purposes.

The third class is for a trivial function of the server used to lookup geolocation codes, and neither requires nor checks any information associated with a user and can therefore be ignored. It cannot possibly leak timing information about users or user-sessions.

Given all of the possible entry points into the server, impersonating a user would require an attacker to know either a valid user-session token, or the password for some username. The presence of a side channel from which a third party could learn information that allows them to impersonate a user would require the server to leak sufficient information about the verification of either the user-session token in the cookie when handling an authenticated request, or the password when handling an unauthenticated request.

A timing side channel in user-session token verification would manifest itself in the lookup method on the user-session tokens structure. This structure is a custom implementation of a hash map (from user-session token [string-representation of a UUID] to Session object). The implementation of the contains() method of the hash map is a null-check on the return of a call to the hash map's get() method. Calls to get() first check for presence of entries in the map who's keys map to the same entry in the hash map's backing array. The algorithm returns quickly in the event of a key miss. A key hit begins a search for a matching key at the corresponding bin position in the hash map. This search boils down to a character-wise comparison of UUIDs across the entries in the particular bin.

Assuming that the timing in this string comparison can be reliably observed, a side-channel attack on user-session tokens (UUIDs) would happen as follows: An attacker would be able to craft UUIDs that hashed to different values to observe hash collisions, and observing which hash bin takes the longest to process. The attacker would begin sending only UUIDs hashing to the known-collision bin, attacking string comparisons early returns on failure. This attack proves impossible in practice because reliable timing data cannot be obtained.

A timing side channel in the password verification algorithm would attempt to observe the timing of the comparison of the hashed password against the hashed password guess. Unfortunately, the implementation of the password check is written to have equal instruction counts per-branch of the comparison code. Said another way, an incorrect password takes the same amount of time to process as the correct password. The only exception to this is a small timing difference when the hashed password and hashed guess are not of equal length. Using this information, an attacker might be able to determine the length of a password, but not the password itself."

Evaluation: Invincea correctly concluded that the challenge application did not contain an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.12.2 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that "the AC vulnerability is the fact that each image filter is a complex time-consuming process. An attacker can add arbitrarily many filters to run on any image, to include duplicates. Specifically, in SnapServiceImpl.modifyFilter() adds filters without checking for duplicates (because it's being added to an ArrayList) using the following code:

```
List<Filter> filters = new ArrayList(photo.getFilters());
if (isAdding) {
  modified = filters.add(filter);
}.”
```

Invincea provided an exploit which demonstrates this vulnerability.

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.13 SubSpace

A.2.9.2.13.1 Question 041 (SC Time, Intended Not Vulnerable, Answered No)

Invincea responded that the application provides a “very narrow scope of interactions, and as such limits the attack surface tremendously. When observing encrypted packets, the attacker is limited to watching a user perform one of the following actions: register a username, confirm a user registration, or update a user's location. Each action will query the database to check that the user exists, but that implementation is via hash lookup of the entire username, so the operation takes constant time regardless of the username length. As the session is encrypted, the attacker must guess what action the user is performing, though it's not unreasonable to assume a specific user will initially register, then confirm, and then subsequent operations will update the user's location. As the username and the password can be of variable length, and since the server does not include information derived from either in its response, determining the length of either is not possible.

Similarly, when directly interacting with the server, the attacker is limited to registering a username, confirming a username, and updating a user's location. The response time for registering a username will be slightly smaller if the username already exists, but only if the attempted username exactly matches the extant username, and the server's response will indicate failure. Confirmation and location updates require the username and password to match the contents of the database, but do not interact with data regarding other users.

As such, there is a slight timing attack in the server, in that attempting to register a username that already exists will take less time than registering a new username, but the server will indicate the success or failure of registration in its response as well. There is no exploitation of this timing attack available, as there's no indication of practical success, so the only approach is a brute force search of the username space, and this will quickly exceed the budget as the username grows. Assuming a strictly lower-case alphabetic username, a four character username will require 26^4 = 456976 attempts in the worst case; the server imposes no such restriction on usernames, however.

Given this, achieving 95% probability of success in 100,000 operations does not seem feasible.”

Evaluation: Invincea correctly concluded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.13.2 Question 042 (SC Space, Intended Vulnerable, Answered Yes)

Invincea responded that “an attacker can exploit the send mail functionality and the fact that it sends mail to the nearest neighbor to triangulate the target user's actual location in sending under 2500 messages to a precision of 1e-4 degrees.”

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.14 TextCrunchr 1

A.2.9.2.14.1 Question 021 (AC Time, Intended Vulnerable, Answered No)

Invincea responded that "question 21 requires an attacker to exploit an algorithmic complexity vulnerability that will cause the target application to run for more than 300 seconds. The attacker has a resource budget of 400000 bytes of input.

Other variants of this target have vulnerable implementations of hashmap functions or sorting functions. This implementation uses a tree node structure that is safely implemented, and only uses a single non-vulnerable sorting function. Profiling the target while running showed no other computationally expensive functions."

Evaluation: Invincea did not identify that the application is vulnerable to a zip bomb. The ZipDecompressor will continuous unzip data from an archive without a limit on the depth of the archive. Given an input zip file that reproduces itself after zipping the resource usage limit can be exceeded.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.15 TextCrunchr 2

A.2.9.2.15.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that "the hashmap implementation for this challenge is different from all the other TextCrunchr programs. Instead of having a Tree, Node based hashmap, textcrunchr_2 uses just a HashMap using a list of Entries. Each entry is a key-value pair. In addition to this list, a set is used to group the keys and a collection is used to group the values.

The worst case of a hashmap is O (n) this can be achieved with the imperfect hashing function which is implemented. An attacker can construct a payload with as many index-hash collisions as possible to reach this worst case time. Collisions in hash-index mappings (where the key hash mod length collides with different hashes mod length) turn the hashmap into a linked list."

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.16 TextCrunchr 3

A.2.9.2.16.1 Question 011 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that "TextCrunchr 3 has a sorter just like the other TextCrunchr applications, but in this version there are two sort functions […] changingSort() and quickSort(). The data first goes through the changingSort algorithm and is then passed to the quickSort function. The partition points of the two merge points look like one should be able to create a data set that, when sorted by changingSort, is problematic for quickSort.

[… Invincea noted that] the pivot point of quickSort is the end element. With quickSort, the worst case occurs when the input array is already sorted - if the pivot is chosen to be the first or

last element. The complexity of this case has a worst case upper bound of O(n^2). This means that the array given to quickSort() must be already sorted, with the last element (the pivot) as the biggest or smallest element in the array. But instead of passing the program this type of pre-sorted array, an attacker needs to pass it an array that takes into account the scrambling done to it by the changinSort() function.

The changingSort() function presorts the list by using a recursive series of 7 midpoints each breaking up the list and sorting the subparts using MergeHelper. However it only does this for sublists after the initial list middle ((start + end) / 2). This means an attacker can submit a worst case list as a sublist (from index 0 to midpoint) of the full text given to the program."

Evaluation: Invincea correctly identified the intended vulnerability in the challenge application and provided a working exploit.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.17 TextCrunchr 5

A.2.9.2.17.1 Question 017 (AC Space, Intended Not Vulnerable, Answered Yes)

Invincea responded that exceeding the resource usage limit "requires large memory size used by the JVM. To do this an attacker can use the stack based sorting algorithm. Specifically a new stack is created by the changingSort() function. An attacker needs to generate input which, inside doIt1(), forces the first branch of the if statement to happen. A pop happens no matter what, but the attacker needs the series of 7 pushes the happen as well. The ArrayIndex must have two things: index.getStart() < index.getEnd() and also index.isPartition()"

Evaluation: This variant of TextCrunchr did not contain any unintended vulnerabilities. Apogee investigated the unintended vulnerability reported using the script provided by Invincea. This script resulted in an "Exceeded file size limit!" message and a "java.lang.RuntimeException: File is too large..." message from the TextCrunchr application. The application then terminated. The exploit provided by Invincea violated the input guards of the challenge application.

Post-Engagement Analysis Ruling: Incorrect

A.2.9.2.17.2 Question 024 (AC Time, Intended Not Vulnerable, Answered No)

Invincea responded that "other variants of this target have vulnerable implementations of hashmap functions or sorting functions. This implementation uses a safely implemented tree node structure for the hashmap functionality, and the sorting functionality, while unconventional, is not vulnerable. Profiling the target while running showed no other computationally expensive functions."

Evaluation: Invincea correctly concluded that the challenge application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.18 TextCrunchr 6

A.2.9.2.18.1 Question 008 (AC Time, Intended Vulnerable, Answered Yes)

Invincea responded that "the AC vulnerability in TextCrunchr 6 is within the SORT function. Invincea input a large set of small random words in ascending order and the doIt functions are executed in order to sort the list. To stay within the input budget, the input file is compressed and presented to the challenge binary as zip file, which it decompresses and processes.

[Invincea's] uncompressed orderedList.txt file has a few properties the small random words allows one to trigger the AC vulnerability many times as it runs over each distinct word, split by spaces. The random words are constructed to be in ascending order as specified by the SORT function's default comparator:

```
 public int doIt0()
  {
   return this.object1.compareTo(this.object2);
  }
```

This means [Invincea's] exploit causes a worst case SORT for each word in the list as the application attempts to place them in descending order. As the exploit list is quite long, the worst case time grows for each word."

Evaluation: The intended vulnerability in the application is within the SORT function. The vulnerability is that if the input list to the function is divisible by 3 the "slow" case of the function is triggered. Invincea's response does not indicate that they identified this vulnerability. The file generated by Invincea contains 100,000 total words. The SORT function is called for unique words and the input file generated by Invincea contains 99,996 unique words which is divisible by 3. Without the happenstance of the input file containing a number of unique words divisible by 3 the exploit would not have worked. It is not clear from Invincea's response that they identified this vulnerability but the decompiled code provided by Invincea shows the vulnerable segment of code which adds more items to the sort stack in the case where "this.list.size() % 3 == 0". This could indicate that Invincea identified the intended vulnerability and purposely crafted an input where the number of unique words is divisible by 3. Invincea provided a working exploit for this vulnerability.

Post-Engagement Analysis Ruling: Correct

A.2.9.2.19 TextCrunchr 7

A.2.9.2.19.1 Question 033 (AC Time, Intended Vulnerable, Answered No)

Invincea responded that "this challenge program contains a computationally vulnerable implementation of an encoding function, in com.ahancock.enigma.EnigmaMachine.encodeLine(). Input to this function must pass through the com.cyberpointllc.stac.zipdecompression.ZipDecompressor.handleInputStream() function, which checks the length of the input and will reject inputs that are too large and would cause excessive run time. This guard prevents exploitation of the vulnerable code. Other algorithmically complex code, the HashMap implementation and the Sort functionality, are safely implemented. Profiling this target while running showed no other computationally expensive functions."

Evaluation: Invincea did not identify that this variant of TextCrunchr uses a hash table which is based on a red black tree. The tree is not balance on puts, therefore the challenge program is vulnerable to DOS. Provided an input size file with enough collisions the resource usage limit can be exceeded.

Post-Engagement Analysis Ruling: Incorrect

A.3 Engagement 3

A.3.1 Overall Trends

The overall accuracy of the Blue Teams was 59% (59% when not including the control team). Of the 117 questions answered, 71 were answered "yes" and 46 were answered "no". Blue teams answered 35 more questions in E3 than in E1; overall accuracy however, decreased from 65% in E1 to 59% in E3. This decrease in overall accuracy was due to a decrease in the accuracy of "yes" responses from E1 to E3. While the accuracy of "yes" responses decreased, the accuracy of "no" responses by Blue teams increased from 25% in E1 to 52% in E3. As expected, this increase in the accuracy of "no" responses by Blue teams corresponded with an increase in total True Negative Rate (TNR) from E1 (14%) to E3 (62%). Total accuracies for both the R&D Teams and the control team decreased from E1 to E3. The control team's True Positive Rate (TPR) increased 14 percentage points from E1 to E3 while their TNR remained constant at 0% for both engagements. In addition, control team's Answers per Analyst Day (AAD) decreased from 1.97 in E1 to .29 in E3. In contrast, the R&D Teams' AAD increased from .97 in E1 to 1.59 in E3 and while their TPR decreased 15.6 percentage points from E1 to E3, the R&D Teams' TNR increased 45 percentage points.

A.3.1.1 Categories of Incorrect Responses

The incorrect responses by the Blue Teams for both AC and SC questions fall into four categories. These categories are not identical in both AC and SC questions but they mirror each other. For AC questions the incorrect Blue Team response categories are:

AC 25. Did not follow directions or violated the question definitions
AC 26. Failed to identify the complexity within the challenge program
AC 27. Failed to understand the input to the complexity control flow of the challenge program
AC 28. Failed to properly determine the complexity bounds of an identified complexity

For SC questions the incorrect Blue Team response categories are:

SC 25. Did not follow directions or violated the question definitions
SC 26. Failed to identify the secret or the secret location within the challenge program
SC 27. Failed to identify the side channel (processing conditioned on the secret value)
SC 28. Failed to properly analyze the side channel strength

An incorrect response can fit into multiple categories e.g. an application contains a side channel and a team identifies a side channel of insufficient strength. This is classified as both SC 3, failure to identify the intended side channel vulnerability, and SC4, failure to recognize that the identified side channel is not sufficiently strong. The resulting distributions of the percentage of incorrect categories for both AC and SC questions are shown in the two tables below.

Table A-42: Engagement 3 AC Responses

	AC 1	AC 2	AC 3	AC 4
Yes	0%	7%	11%	25%
No	0%	25%	21%	11%
Total	**0%**	**32%**	**32%**	**36%**

Table A-43: Engagement 3 SC Responses

	SC 1	SC 2	SC 3	SC 4
Yes	4%	14.8%	3.7%	59%
No	0%	3.7%	14.8%	0%
Total	**4%**	**18.5%**	**18.5%**	**59%**

For AC questions, no incorrect responses were due to a failure to follow directions or a violation of the question definitions. Failure to properly determine the complexity bounds (AC 4) accounted for the largest percentage of incorrect AC responses; however, this category of incorrect AC responses only accounted for 4 percentage points more incorrect responses than failures to identify the complexity (AC 2) and failure to understand the input to the complexity control flow (AC 3). The main difference in these categories is in the distribution of incorrect "yes" and "no" responses. For AC 2 and AC 3, incorrect "no" responses accounted for more incorrect "yes" responses. This makes intuitive sense given that a failure to identify a complexity or understand the input to the complexity control flow would be more likely to lead a team to conclude that a challenge program does not contain an AC vulnerability. In contrast, incorrect category AC 4 had more "yes" responses than "no" responses. This indicates that teams are more likely to incorrectly identify a challenge application as vulnerable to an AC vulnerability if they are unable to properly determine the complexity bounds of an identified complexity.

In E3, 4% of incorrect SC responses were due to a failure to follow directions or a violation of the question definitions. While the percentage of incorrect responses in this category has decreased from E1 (40%), it is the same as for E2. Failure to properly analyze side channel strength (SC 4) continues to dominate the reason for incorrect SC responses in E3. In E1, this category accounted for 61% of incorrect SC responses, in E2, 60%, and in E3, 59%. This is arguably the most difficult component of evaluating applications for side channel vulnerabilities, therefore the dominance of this category of incorrect SC responses is understandable. In E3, all SC 4 incorrect responses were "yes" responses, meaning that a failure to properly analyze side channel strength results in teams misclassifying a non-vulnerable application as vulnerable. Failure to identify the secret and secret location (SC 2) and failure to identify the side channel (SC 3) each accounted for 18.5% of incorrect SC responses. These categories are mirrored in the distribution of "yes" and "no" responses with "yes" responses dominating in SC 2 and "no" responses dominating in SC 3. The pattern in SC 2 is consistent with Blue Teams being more likely to declare an application vulnerable after identifying a secret location (different from the intended vulnerability) and improperly evaluating the side channel strength. While the pattern in SC 3 is consistent with Blue Teams identifying the secret location of an intended side channel

vulnerability, being unable to identity the side channel and therefore misclassifying the application as non-vulnerable. In E2, SC 2 accounted for 0% of incorrect SC responses, but in E3, SC 2 and SC 3 evenly accounted for incorrect SC responses.

A.3.1.2 Attempted Questions

During E3, teams were asked to log cases where a question was attempted but a response was not provided. Only three teams: UC Boulder, GrammaTech, and Invincea reported attempted questions – cases where they attempted a question without providing a response. In total, there were 33 attempted questions and Invincea accounted for 61% of these cases. 52% of the attempted questions were AC Time questions while SC Space and Time questions accounted for the smallest percentage of these questions, 6%. The table below shows the data for attempted questions in E3.

Table A-44: Engagement 3 Team Accuracy

Team	Attempted Questions	AC Space	AC Time	SC Space	SC Time	SC Space & Time	Team Percentage
UC Boulder	11	2	6	1	2	0	33%
GrammaTech	2	1	0	0	0	1	6%
Invincea	20	1	11	2	5	1	61%
Researchers	13	3	6	1	2	1	
All Teams	33	4	17	3	7	2	
	Percentage of Total	12%	52%	9%	21%	6%	

A.3.1.3 Identified Unintended Vulnerabilities

In E3, Blue Teams identified 9 unintended vulnerabilities that impacted 17 challenge questions; 6 of these questions were intended non-vulnerable and were changed to vulnerable. In E2, Blue Teams identified 2 unintended vulnerabilities that impacted 2 questions, and in E1 Blue Teams identified 2 unintended vulnerabilities that impacted 8 questions. The number of identified unintended vulnerabilities in E3 is a significant increase from the previous engagements, one of which was a take-home engagement. Research teams identified 8 of the 9 E3 unintended vulnerabilities, and the control team and a R&D Team both identified the remaining unintended vulnerability. Northeastern university identified 7 of the 9 unintended vulnerabilities; 6 other R&D Teams identified some of the same unintended vulnerabilities as Northeastern and the 2 remaining unintended vulnerabilities. Given the time spent by the EL and AC teams to evaluate the challenge programs for unintended vulnerabilities prior to the engagement, the ability of the Blue Teams, in particular the R&D Teams, to identify a large number of unique unintended vulnerabilities during a live engagement reflects well on their tools.

A.3.2 University of Colorado Boulder

A.3.2.1 UC Boulder Overview

UC Boulder responded to 7 questions with 29% accuracy. UC Boulder responded to 6 side channel questions and 1 algorithmic complexity question.

UC Boulder's accuracy decreased from their E1 results from 30% to 29%. The tables below show details of UC Boulder's raw accuracy and their accuracy following the initial review.

Table A-45: Engagement 3 UC Boulder Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	1	100	1	100
SC Time	0	0	NA	0	0
SC Space/ Time	0	0	NA	0	0
AC in Space	2	0	0	0	0
AC in Time	4	1	25	1	25
Total	**7**	**2**	**29**	**2**	**29**

Table A-46: Engagement 3 UC Boulder Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	6	1	**17**	1	**17**
Not Vulnerable	1	1	**100**	1	**100**
Yes Answer	1	1	**100**	1	**100**
No Answer	6	1	**17**	1	**17**

A.3.2.2 UC Boulder Specific Responses

A.3.2.2.1 Airplan 2

A.3.2.2.1.1 Question 020 (AC Time, Intended Vulnerable, Answered N/A)

Response: Colorado responded that their *tool suggested that they may be a vulnerability related to a hash function. However; we were unable to run the application after quite some effort... hence unable to test it.*

Evaluation: The challenge program contained an AC Time vulnerability in the sort method implementation. This method implements merge sort, but for input lists with a length divisible by 6, unnecessary merges occur resulting in an AC Time vulnerability. Colorado did not provide a response to this question but noted that they had difficulty running the application during the engagement.

Post-Engagement Analysis Ruling: None

A.3.2.2.2 Airplan 3

A.3.2.2.2.1 Question 025 (AC Time, Intended Vulnerable, Answered N/A)

Response: Colorado responded that their *tool suggested that they may be a vulnerability related to a hash function. However; we were unable to run the application after quite some effort... hence unable to test it.*

Evaluation: The challenge program contained an AC Time vulnerability in the shortest path algorithm. The challenge program uses an implementation of Dijkstra's algorithm which allows negative costs. An input route map with negative cost values can be used to trigger an AC Time vulnerability. Colorado did not provide a response to this question but noted that they had difficulty running the application during the engagement.

Post-Engagement Analysis Ruling: None

A.3.2.2.3 Airplan 4

A.3.2.2.3.1 Question 009 (AC Time, Intended Vulnerable, Answered N/A)

Response: Colorado responded that their *tool suggested that they may be a vulnerability related to a hash function. However; we were unable to run the application after quite some effort... hence unable to test it.*

Evaluation: The challenge program contained an AC Time vulnerability in the shortest path algorithm. The challenge program uses an implementation of Dijkstra's algorithm which allows negative costs. An input route map with negative cost values can be used to trigger an AC Time vulnerability. Colorado did not provide a response to this question but noted that they had difficulty running the application during the engagement.

Post-Engagement Analysis Ruling: None

A.3.2.2.4 Collab

A.3.2.2.4.1 Question 018 (AC Time, Intended Not Vulnerable, Answered Yes)

Response: Colorado responded that *whenever the client modifies his database by adding a new event it is recorded in a log file. A new log file is created with every UNDO (Discarding of the scheduling sandbox calendar). Hence a user can send a sequence of inserts and undos to launch a DOS attack. An exploit is included with this answer.*

Evaluation: The challenge program did not contain an intended AC Time vulnerability. Colorado's exploit uses the input budget to fill the challenge program's receive queue for the assigned port. This occurs when 1378 requests of the sequence init sandbox, insert key, discard sandbox are made. For the 1379^{th} request made, timing is done from the last request sent to the response received from the server. Since the server does not receive the request, a response is never received and by the current operational definition it is an out-of-scope AC Time vulnerability since the receive queue size is not controlled by the bytecode.

Post-Engagement Analysis Ruling: Correct

A.3.2.2.4.2 Question 028 (SC Space, Intended Not Vulnerable, Answered No)

Response: Colorado responded that *there is no side-channel in space. We tried a combination of binary search and observed the wireshark output and it did not show any side channel.*

Evaluation: The challenge program did not contain an intended SC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.2.2.4.3 Question 042 (SC Time, Intended Not Vulnerable, Answered N/A)

Response: Colorado responded that *we suspect that there is no side channel in time as our analysis shows that there is no significant difference in execution time even when the range searched contains the audited event.*

Evaluation: The challenge program contained two unbalanced paths which balanced each other. There was no intended SC Time vulnerability in the challenge program; however, Colorado provided a response indicating their line of thought but did not provide an answer for this question.

Post-Engagement Analysis Ruling: None

A.3.2.2.5 Info Trader

A.3.2.2.5.1 Question 002 (AC Time, Intended Vulnerable, Answered No)

Response: Colorado responded that *both DocumentStore.loadDoc() and SiteMapGenerator::serializeTree() are recursive methods whose depths are not guarded. As a result, feeding the server self-referential documents could easily put it into an infinite recursion. However, this will cause a StackOverflowException to be thrown instead of hung up the server. It may be possible to prevent SiteMapGenerator::serializeTree() from throwing the exception as it only checks the filenames in a case-insensitive manner, we neither came up with any way to exploit this, nor any way to turn other instances of infinite recursion into availability vulnerabilities.*

Evaluation: The challenge program did not contain an intended AC Time vulnerability. Northeastern identified an unintended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Incorrect

A.3.2.2.5.2 Question 004 (AC Space, Intended Vulnerable, Answered No)

Response: Colorado responded that *the server tries to serialize the uploaded document onto its disk, and the size of the a forementioned serialization is controlled by InfoTraderWriter.emit(). Our analysis did not find any loops in this part of the codes that an attacker could inflate in a super-linear manner. We breifly examined the codes manually but also did not find any suspicious behavior.*

Evaluation: The challenge program contained a broken input guard which while intended to prevent cycling allowed users to upload a document with the same name as the root node. If an attacker uploads a document linking to the document with the same name as the root, the resulting document traversals that generate the sitemap xml file will not cause a StackOverflowError but will result in a sitemap xml file which exceeds the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.3.2.2.6 Linear Algebra Platform

A.3.2.2.6.1 Question 012 (AC Time, Intended Vulnerable, Answered No)

Response: Colorado responded that *the matrix multiplication appears to be the most costly procedure in this application. The budget limits the size of matrix multiplication to two 600x600 matrices. This results in a runtime of a minute, which is below the target.*

Evaluation: Colorado correctly identified that the matrix multiplication operation was the most computationally expensive user operation. Colorado did not recognize that the user can control the distribution of sub-tasks in computing the matrix product and trigger an AC Time vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.3.2.2.7 Malware Analyzer

A.3.2.2.7.1 Question 031 (AC Time, Intended Not Vulnerable, Answered N/A)

Response: Colorado responded that *we observed strange behavior regarding the timing, but we were not able to translate that into an exploit. Our regex analyzer indicates that none of the regexes are exploitable.*

Evaluation: The challenge program did not contain an intended AC Time vulnerability. Colorado provided an explanation of their thoughts but not a response to the question.

Post-Engagement Analysis Ruling: None

A.3.2.2.7.2 Question 035 (AC Space, Intended Vulnerable, Answered No)

Response: Colorado responded that their *analysis tool CodeHawk indicates that the DasmHelpers.make_function_cfg function produces the largest log file. We have been trying to produce the worst case, which appears to be a dasm file as input with one call and only conditional jumps. Within the budget of 3000 bytes, this produces a log file of 11000 bytes which is less than the 80000 bytes target.*

Evaluation: The challenge program contained an intended AC Space vulnerability in the packer detector functionality. Given a 256-integer feature vector, while calculating the average value heuristic an attacker can bypass the input guard to trigger an arithmetic exception. This exception is written to the log file. Normal log writes are overwritten, but the exception causes the entire stack trace to be appended to the log file.

Post-Engagement Analysis Ruling: Incorrect

A.3.2.2.8 Tweeter

A.3.2.2.8.1 Question 049 (AC Time, Intended Vulnerable, Answered No)

Response: Colorado responded that they *obtained a list of high complexity methods with CodeHawk. Most of these methods are associated with either the com.tweeter.service or vash.operation packages. CodeHawk indicates that most of the loops in com.tweeter.service are either bounded with small constants (3, 27) or by the size of the tweet. The loops in the high complexity methods in vash.operation are generally bounded by the height or width of the ImageParameter object they receive, and these values are fixed by Tweeter at 128 which are again not large enough to produce the desired runtime.*

Evaluation: The challenge program contained an AC Time vulnerability in the spell-correcting feature. The spell check algorithm's runtime is a function of word length and number of mistakes; however, the application has a maximum for word length and for number of mistakes. By providing a word which is within the applications maximum word limit (26 characters) and contains just enough mistakes to trigger the spell check algorithms slow execution time, the resource usage limit can be exceeded.

Post-Engagement Analysis Ruling: Incorrect

A.3.3 Draper Labs

A.3.3.1 Draper Labs Overview

Draper responded to 7 questions in Engagement 3 with an 71% overall accuracy and an 80% accuracy in "No" responses. Draper's accuracy in this engagement was greater than their Engagement 1 and 2 accuracies answering more questions as well.

Table A-47: Engagement 3 Draper Labs Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	1	100	1	100
SC Time	2	2	100	2	100
SC Space/ Time	0	0	NA	0	NA
AC in Space	1	0	0	0	0
AC in Time	3	2	67	2	67
Total	**7**	**5**	**71**	**5**	**71**

Table A-48: Engagement 3 Draper Labs Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	2	1	**50**	1	**50**
Not Vulnerable	5	4	**80**	4	**80**
Yes Answer	2	1	**50**	1	**50**
No Answer	5	4	**80**	4	**80**

A.3.3.2 Draper Labs Specific Responses

A.3.3.2.1 Collab

A.3.3.2.1.1 Question 018 (AC Time, Intended Not Vulnerable, Answered Yes)

Response: Draper responded that *Starting from scratch (event.data is unpopulated), we can create a denial of service by exploiting bad computational behavior of the 'SchedulingSandbox.add' method following the sequence of expensive calls, which we determine using the taint analysis with the 'packet' arguement of channelRead0:*

collab.CollabServer.channelRead0(ChannelHandlerContext;DatagramPacket;)V:613
collab.SchedulingSandbox.populateSandbox(EventResultSet;)SchedulingSandbox;:48
collab.SchedulingSandbox.<init>([I)V:6
collab.SchedulingSandbox.add([I)V:19
collab.SchedulingSandbox.add(IDataHolder;)Z:2
collab.SchedulingSandbox.addhelper(I)V:11
collab.SchedulingSandbox.addhelper(DataNode;I)V:7
collab.SchedulingSandbox.addhelper(I)V:17
collab.SchedulingSandbox.addhelper(DataNode;I)V:189
collab.SchedulingSandbox.split(DataNode;)V:200
collab.SchedulingSandbox.replace(TempIndexNode;DataNode;DataNode;DataNode;)V:50
collab.SchedulingSandbox.replace(TempIndexNode;DataNode;DataNode;DataNode;)V:174
collab.SchedulingSandbox.sort(...)

The complexity analysis determines that 'addhelper', 'split' and 'sort' are potentially expensive function, in addition 'addhelper' is recursive. To exploit this behavior, we need to avoid the artificial limitation in 'DataNode.takestep' which prevents too many 'ADD' queries in succession, but once the database is populated, a few hundred calls to ADD will suffice.

Evaluation: The challenge program does not contain an intended AC Time vulnerability. The potential vulnerability in the add method identified by Draper was investigated prior to the engagement and found to be insufficient to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.3.3.2.1.2 Question 028 (SC Space, Intended Not Vulnerable, Answered No)

Response: Draper responded that they *believe that the outstream data is independent of the value of the 'CollabServer.auditids' field. In particular the 'bos' variable of the 'CollabServer.channelRead0' method contains the server response.*

The flow for constructing this response splits into 2 cases:

*Either the sandbox is *not* initialized, in this case the response is populated by 'btreeformastercalendar.searchR', which does not contain the audits at all.*

Otherwise the sandbox is filled with everything including audits by 'SchedulingSandbox.popluateSandbox', and the response is computed using the 'SchedulingSandbox.getRange' method, which in turn calls

TempIndexNode.takeStep -> DataNode.takeStep

the last of which excludes non normal user data.

Evaluation: The challenge program does not contain an intended SC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.3.2.1.3 Question 042 (SC Time, Intended Not Vulnerable, Answered No)

Response: Draper responded that *the audits are stored in a Map in the class CollabServer, which maps usernames to audit-IDs. Since the values of the Map are audit-IDs (as opposed to*

Collections of audit-IDs) there can be at most one audit per user. When the sandbox is initialized on the server, all appointments -- including the audit -- are written to a map from event-IDs to additional information associated with the event. So if the sandbox was initialized before sending a search query, then in theory there may be a difference in timing behavior as the audit-ID has to be filtered while the result is constructed (which is happening in DataNode.takestep). In this way, an attacker could infer if a given range contains an audit-ID or not. However, as there is just a single audit ID, the resulting difference in timing is extremely small and all of our attempts to measure it failed. We tried the following attacks:

** comparing the time required to search equally sized ranges with and without audit-ID*
** comparing the time required to search ranges containing equally many appointments with and without audit-ID*

In both cases, we tried differently sized ranges. However, we weren't able to measure a significant difference in timing behavior. Apart from search querys, we don't see another potential way for an attacker to infer information about audit-IDs. Additionally, the 'SchedulingSandbox.getRange' function which runs when the sandbox is active does not depend on the min and max range of the query, which prevents doing a binary search on query ranges, even if there were time differences.

Evaluation: The challenge program does not contain an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.3.2.2 Malware Analyzer

A.3.3.2.2.1 Question 031 (AC Time, Intended Not Vulnerable, Answered No)

Response: Draper responded that their *complexity analysis identifies several methods with potential high resource usage:*

com.ainfosec.MalwareAnalyzer.Server.createChecksum([B)[B -> Unknown Complexity
com.ainfosec.MalwareAnalyzer.Server.serve(Lfi/iki/elonen/NanoHTTPD$IHTTPSession;)Lfi/iki/elonen/NanoHTTPD$Response; -> Unknown Complexity
com.ainfosec.MalwareAnalyzer.DasmDatabase.compute_all_similarities(Ljava/lang/String;)Ljava/util/NavigableSet; -> Quadratic Complexity
com.ainfosec.MalwareAnalyzer.DasmHelpers.make_function_cfg(Ljava/util/TreeMap;Ljava/lang/Integer;)Lcom/ainfosec/MalwareAnalyzer/CFG; -> Unknown Complexity

The other functions are either recursive or determined to be constant or linear.
The only interesting recursive function we identified is the 'Analysis.compute_cosine_similarity'

Of these, the serve function is the most complex. One of the cases of the switch function prints a representation of the whole call graph, which may be quadratic in the number of functions.

However, it is unlikely to be the source of a vulnerability: the worst case is a program in which every function calls every other function. But this requires a quadratic size in the number of functions to write. We do not believe that there is a way to exceed the resource usage limit using fewer than 3000 bytes.

Evaluation: The challenge program did not contain an intended AC Time vulnerability

Post-Engagement Analysis Ruling: Correct

A.3.3.2.2.2 Question 035 (AC Space, Intended Vulnerable, Answered No)

Response: Draper responded that *here only one method writes to file: 'Server.serve()' (via the 'Logger.write_log_entry' method). The analysis is similar to the time analysis, where the output is largest for the 'get_cfg' requests. The log file is overwritten at each query.*

We do not believe that there is a way to get output incommensurable with the size of applications added with 'add_dasm'. In particular, we do not believe that it is possible to get 80 KB of output starting with an empty database and an input of 3000 bytes.

Evaluation: The challenge program contained an intended AC Space vulnerability in the packer detector functionality. Given a 256-integer feature vector, while calculating the average value heuristic an attacker can bypass the input guard to trigger an arithmetic exception. This exception is written to the log file. Normal log writes are overwritten, but the exception causes the entire stack trace to be appended to the log file.

Post-Engagement Analysis Ruling: Incorrect

A.3.3.2.3 RSA Commander

A.3.3.2.3.1 Question 033 (AC Time, Intended Vulnerable, Answered Yes)

Response: Draper responded that *there is a "while" loop in 'stac.communications.Communications.doHandshake which may run *forever* (in particular more than 2 hours) if the result of 'HandshakeHandler.handle' always returns HANDLER_STATE.WAITING*

For this to happen, the stac.communications.Handler.handle methods needs to always reteren HANDLER_STATE.WAITING. This happens if the branch depending on 'isPacketStillOK(PacketBuffer)' is hit and returns true, isPacketStillOK always returns true for instances of 'RequestHandler'.

The branch is reached if:
*isPacketFormed(PacketBuffer) is *false**

Other conditions which must be true (otherwises an infinite loop occurs)
is.available() == 0 (no bytes can be read)
OR
is.read() == -1 (end of the stream)

Call graph to sendMessage is:
stac.Client.run() -> stac.Client.Console.run() -> stac.client.Console.handleUserInput() -> stac.communications.Communications.sendMessage()

Crucial point:
*RequestHandler.isPacketFormed needs to return *false* on a packet built from the remote input*

This seems easy to do:
*In particular, *not* responding seems to be sufficient!*

The complexity analysis tool identifies 'stac.communications.Communications.sendMessage' as a method with potential infinite running time,

The taint analysis shows that the address of the attacker can potentially reach the 'stac.communications.Handler.handle' InputStream argument, which on inspection is controlled in a strong way by the remote client.

A script (in ruby) which reproduces the attack is:
require 'socket'
TCPServer.open(8081)

On the victim side, just running any client and sending any message to port 8081 will result in an infinite loop.

Evaluation: The challenge program contains and AC Time vulnerability in decrypting sent messages. An attacker can cause a benign user's instance of the application to trigger the slow mode of decryption by setting the counter value to 0. Northeastern and Draper identified an unintended vulnerability in the handshake procedure of the challenge program. When userA is attempting to send a message to userB, userA initiates the handshake procedure with userB. During the handshake procedure, the Communications.doHandshake() method calls the HandshakeHander.handle() method with a while loop that only terminates when HandshakeHander.handle() != HANDLE_STATE.WAITING. HandshakeHandler.handle() returns HANDLER_STATE.WAITING if isPacketStillOK() returns true. As implemented in HandshakeHandler.isPacketStillOK(), this method always returns true. If userB provides a partial response to userA's handshake request, the Communications.doHandshake() method will be stuck waiting for the completion of the message. While the victim user must initiate communication with the attacker, this represents an AC Time vulnerability by the strictest interpretation of the current operational definition

Post-Engagement Analysis Ruling: Correct

A.3.3.2.3.2 Question 040 (SC Time, Intended Not Vulnerable, Answered No)

Response: Draper responded that t*he message to send is (also) processed and encrypted in 'Communications.sendMessage' simply using the call-graph and reading the decompiled code, we can find that the succession of methods used to process each byte of the message are, in order*

Communications.sendMessage ->
-> RequestPacketParser.serialize
-> RequestPacketParser.appendRijndaelEncryptedSessionContent
-> DES.encrypt_ctr
-> DES.encrypt
-> DES.encryptBlock
-> DES.getBytesFromLong

*For each of these methods, we can use the run-time upper *and lower* bounds to see if there seems to be a discrepancy between the longest and shortest run-time for these methods. Note that if a discrepancy exists, we have no guarantee of a vulnerability or vice-versa.*

However, most methods are seen to have identical bounds, meaning that the run-time is not sensitive to the input, except for the 'getBytesFromLong' method.

However, we do not believe that the time behavior of this method is sufficient to give information about the bytes from the original method.

We conclude that there is no side channel vulnerability which reveals the message, but without formal guarantee.

Evaluation: The challenge program does not contain an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.4 GrammaTech

A.3.4.1 GrammaTech Overview

GrammaTech responded to 9 questions with an 56% accuracy. GrammaTech attempted several side channel questions but focused primarily on algorithmic complexity questions. GrammaTech answered 3 AC Space questions with 100% accuracy.

Table A-49: Engagement 3 GrammaTech Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	3	1	33	1	33
SC Time	0	0	NA	0	NA
SC Space/ Time	0	0	NA	0	NA
AC in Space	3	3	100	3	100
AC in Time	3	3	100	1	33
Total	**9**	**7**	**78**	**5**	**56**

Table A-50: Engagement 3 GrammaTech Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	7	7	**100**	5	**71**
Not Vulnerable	2	0	**0**	0	**0**
Yes Answer	9	7	**78**	5	**56**
No Answer	0	0	**NA**	0	**NA**

A.3.4.2 GrammaTech Specific Responses

A.3.4.2.1 Airplan 1

A.3.4.2.1.1 Question 052 (AC Time, Intended Vulnerable, Answered Yes)

Response: GrammaTech responded that *the class "ConnectedAlg.java" contains a vulnerable method "isConnected" that is invoked every time the user queries properties of the flight map. ore specifically, this function is responsible to check that a queried flight map is strongly connected. It is also invoked for finding an Eulerian path for a node (e.g., an airport) in the flight map. From visual inspection, this function appears to have cubic complexity in the number of nodes in the graph. It also calls some utility functions such as "getNextComplementaryEdgeId" in the class "SparseIdFactory.java" and "validateGraph" in the class "Accommodation.java" in order to check the connectivity of the given graph.*

Why vulnerable code can be triggered by an attacker:

We crafted an attack by creating a map containing a large cycle. The map is a graph with 500 nodes; node 0 is connected to node 1, node 1 to node 2, and so on; finally, node 499 is connected back to node 0. Thus checking connectedness requires traversing the entire graph. The file containing this flight map is only 17.8K in size and satisfies the input budget.

How their *tools supported discovery of the vulnerability:*

Our tools identified isConnected() as high-complexity code. It also identified CalculateOptimalTrail.calculateOptimalTrail() as high-complexity code - this method is called when the users query a shortest path from one airport to a destination airport. Profiling confirmed that checking connectedness and related computations take up a significant amount of CPU time. Based on these observations, we crafted our attack.

Evaluation: The challenge program contains an AC Time vulnerability in its shortest path algorithm. Given a route map, the application calculated the shortest path using A-star search with an inconsistent heuristic. GrammaTech identified a potential vulnerability in the *isConnected* method and provided an input graph containing a large cycle. GrammaTech's exploit was tested with a 500 airport route map with a large cycle Using the text file format we were able to stay under the input budget (25000 bytes) with 12061 bytes of input submitted. When the properties request was submitted, the server provided a response in less than 2 seconds. The response stated that for all k-connectedness parameters (2,3,4, and 5), the connectedness was "Too large to tell". The challenge program in the edu.cyberapex.flightplanner.GraphTranslator.kConnected() method does not compute the connectedness for input graphs of greater than 400 airports. We modified the attack for 400 airports and this resulted in a resource usage of 52 seconds. This is under the resource usage limit of 500 Seconds. GrammaTech however identified and reported a vulnerability in the shortest path implementation of the challenge program.

Post-Engagement Analysis Ruling: Correct.

A.3.4.2.2 Airplan 2

A.3.4.2.2.1 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Response: GrammaTech responded that *the classes "KXmlRetriver.java" and "ConnectedAlg.java" contains vulnerable methods "readValue" and "isConnected", respectively.*

Whenever the users want to add a new flight map and query about the properties of the map, these functions are invoked.

Why vulnerable code can be triggered by an attacker:

We crafted an attack by adding the three large provided flight maps "many_airports_map.txt" (1.9KB), "may_flights_map.txt" (9.4KB), and "test_map500.txt" (12KB). It satisfies the size limit of the request sent to the server (at most 25KB).

How their *tools supported discovery of the vulnerability:*

Our tools identified readValue() and isConnected() as high-complexity code based on the number of complexity sinks contained. Profiling confirmed that those functions and their callees such as "grabAirports" "pullAirpots" and "getAirport" consume significant execution time. Based on those observations, we crafted our attack.

Evaluation: The intended vulnerability in the challenge program was in the sort method. Provided a list of n items where n is divisible by 6, the implementation of merge sort performs many unnecessary merges resulting in an AC Time vulnerability. GrammaTech identified a potential vulnerability in the *isConnected* method and provided an input graph containing a large cycle. GrammaTech's exploit was tested the three example route maps, none of which exceeded the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.3.4.2.3 Airplan 3

A.3.4.2.3.1 Question 025 (AC Time, Intended Vulnerable, Answered Yes)

Response: GrammaTech responded that *the classes "KXmlRetriver.java" and "OptimalPath.java" contains vulnerable methods "readValue" and "pullPathVerties", respectively. Whenever the users want to add a new flight map the method "readValue" is invoked, while "pullPathVerties" is invoked during the process of finding the optimal (shortest) path with respect to the type of cost (e.g., fuel cost, distance).*

Why vulnerable code can be triggered by an attacker:

The attackers can always generate maps with large maps for querying the shortest path from a particular pair of airports. For example, we crafted an attack using two maps: "test_map400.txt" (9.6KB) and "test_map500.txt" (12KB). The graph for "test_map400.txt" is a simple graph with height is 1 with 400 nodes. The number of edges is 399, from node i to 000, where i = 001,...,399.
The graph for "test_map500.txt" is similar. Botn maps together satisfy the size limit of the request sent to the server (at most 25KB). We then ask the server to find the optimal (shortest) path from the "final airport" (id 399) to the "first airport" (id 000) w.r.t the distance in the "test_map400.txt" map and to find the shortest path from the "final airport" (id 499) to the "first airport" (id 000) w.r.t the fuel cost in the "test_map500.txt" map.

How their *tools supported discovery of the vulnerability:*

Our tools identified "readValue" and "pullPathVertices" as potential high-complexity code; with profiling we confirmed that these methods and their callees consume significant execution time. Based on those observations, we crafted our attack.

Evaluation: The challenge program contained an intended vulnerability in calculating the flight crew required for a given route map. This is done by solving the max flow problem using the Ford Fulkerson algorithm. Given an input graph with rational capacity values, the resource usage limit can be exceeded. The challenge program did not contain an intended AC Time vulnerability in its shortest path implementation. The inputs used by GrammaTech are less than the input budget; however, when the exploit was attempted resource usages of 8.577 and 16.287 for the uploads of *test_map500.txt* and *test_map400.txt* respectively were observed. The resource usages observed for the shortest path commands were 2.9 and 4.5 seconds for *test_map500.txt* and *test_map400.txt* respectively. This exploit does not exceed the resource usage limit of 500 seconds.

Post-Engagement Analysis Ruling: Incorrect

A.3.4.2.4 Linear Algebra Platform

A.3.4.2.4.1 Question 005 (AC Space, Intended Vulnerable, Answered Yes)

Response: GrammaTech responded that *LinearAgebraService.serve calls ShortestPathOperation.compute calls readMatrixFromCSV with unsafe arguments.*
In the class ShortestPathOperation, the function call ms2.readMatrixFromCSV(..., false, false) does not check that its arguments are within bound. Then it proceeds to allocate a matrix of arbitrary size, as determined by user input.

Why vulnerable code can be triggered by an attacker:

We directly exploited the above vulnerability to create an attack that exceeded the space budget.

How their *tools supported discovery of the vulnerability:*

Our initial heuristics identified readMatrixFromCSV() as a potential vulnerability; we confirmed this via manual inspection.

Evaluation: The challenge program does not contain an intended AC Space vulnerability. GrammaTech's response is the same as the one provided by Utah, essentially the challenge program uses the MatrixSerializer.readMatrixFromCSV() method to parse input matrices. This method has two enforcement parameters enforceWidth and enforceSize which are used all but one of the possible input matrices. The shortest path operation of the matrix takes two input matrices, an adjacency matrix and a targetNodes vector representing the set of nodes to find the shortest path between. The vector of nodes is parsed with the enforceWidth and enforceSize parameters set to false. The readMatrixFromCSV method proceeds to allocate memory for the targetNodes vector using the user submitted arguments which results in a memory usage of 3,653.928 MB exceeding the resource usage limit of 2,500 MB. An illegal argument exception is then triggered once the method finds that the number of rows in the input matrix and the number of rows declared by the user do not match; however, this is after the memory has been allocated for the targetNodes vector. This is an in-scope unintended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.4.2.5 Malware Analyzer

A.3.4.2.5.1 Question 035 (AC Space, Intended Vulnerable, Answered N/A)

Response: GrammaTech responded that they *attempted to use* our *toolchain to prove that there was no complexity vulnerability in space, but we were not able to do that during the timeframe of the live engagement. Our approach to this question was to first create a model of the malware analyzer's Server.serve() method, and then to analyze this model with our tools. The model abstracts the original Java code to an integer program representing the size of the Server.response variable and a few other string variables. Although we currently need to construct this model by hand, we hope to be able to construct a model of this kind automatically, or in a partially-automated way, in the future. If we had been able to prove an upper-bound on the size of the Server.response variable, then we would have concluded that there was no complexity vulnerability in space in malware_analyzer, with some qualifications asserting that our model was a sound abstraction of the original program. However, during the live engagement, our tool was not able to prove such an upper-bound.*

Evaluation: The challenge program contained an intended AC Space vulnerability in the packer detector functionality. Given a 256-integer feature vector, while calculating the average value heuristic an attacker can bypass the input guard to trigger an arithmetic exception. This exception is written to the log file. Normal log writes are overwritten, but the exception causes the entire stack trace to be appended to the log file.

Post-Engagement Analysis Ruling: None

A.3.4.2.6 SmartMail

A.3.4.2.6.1 Question 039 (SC Time/Space, Intended Vulnerable, Answered N/A)

Response: GrammaTech responded that they *attempted this question and identified some suspicious code, but were unable to definitively answer whether a side channel exists within the time limits of the engagement.*

Our automated tools, and in particular our SCL detection heuristics, identified smartmail.process.controller.module.EmailSessionReducer.reduce() as a method containing conditionals with different execution costs on different branches. Visual inspection confirmed that this method makes an extra logging call if the value it is processing is a secret email address. Thus there is potential for a SCL.

We confirmed that the above reduce() method is invoked whenever executing smartmail.process.controller.module.PipelineController.main(), which is invoked in smartmail.messaging.controller.module.EmailSenderReceiver.doEmail(). This method is invoked whenever a user performs a POST request to send email via the server; it is therefore reachable from user (and attacker) input.

However, we were unable to determine whether an SCL exists within the time limits of the engagement. Specifically, we were unable to:
- identify an execution path that would actually cause the logging code to be executed; modeling the dataflow associated with a map reduce invocation is something our tools currently do not support.

Evaluation: The challenge program contained an SC Time/Space vulnerability which leaked the administrator of a given mailing list. The vulnerability was in the log file writes that occur when a message is sent. The set of all recipients of a given message (which includes the mailing list administrator and each of the subscribers) is alphabetized and written to a log file with the mailing list subscribers written to one log file and the administrator written to another. Using this side channel an attacker can run a binary search for each character and determine each subsequent character of the administrator's email beginning at the first character. GrammaTech identified the conditional branch within the code's logging functionality for *a secret email address*. However, GrammaTech did not identify the equivalent logging functionality for non-secret email addresses. This information could have been used to identify the side channel within the challenge program. Given enough time, it appears GrammaTech could have identified the vulnerability within the challenge program. While they did not provide a response to this question, the provided detail of their work on the question indicates that they were on the correct path to identifying the intended vulnerability.

Post-Engagement Analysis Ruling: None

A.3.4.2.7 WithMi 1

A.3.4.2.7.1 Question 010 (SC Space, Intended Vulnerable, Answered Yes)

Response: GrammaTech responded that *the method transmitMessage in edu.networkcusp.chatbox.HangIn has a loop over the number of participants (which is the secret). For each participant, it builds and sends a different message. The size of each message will be similar for all participants in the chat. The number of packets will also be linear in the number of participants.*

Side channel strength and triggerability by an attacker:

For this question, we assume that the attacker is able to watch all packets in the network, including packets that are sent between users in the chat he is not invited to. From the above discussion of the functionality of transmitMessage, it appears that the attacker can determine the number of participants in the chat if he can observe those packets.

How their *tools supported discovery of the vulnerability:*

This was determined by manual inspection of the decompiled source code.

Evaluation: GrammaTech identified the intended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.4.2.7.2 Question 053 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: GrammaTech responded that *transmit and transmitGuide in edu.networkcusp.chatbox.FileTransfer are responsible for sending a file. There is a linear relation between the size of the compressed file and the number and size of the packets that are sent. Each packet contains 256 bytes of the file (except the last one that contains less), and has some extra data of constant size. Then, there is a linear relation between the size of the compressed file and the number (and size) of packets sent.*

Side channel strength and triggerability by an attacker:

From the above discussion, it appears that an attacker should be able to infer the size of the compressed file by looking at the packets, and thus identify what file has been sent if he can try to compress the files manually to see their compressed sizes.

How their *tools supported discovery of the vulnerability:*

This was determined by manual inspection of the decompiled source code.

Evaluation: GrammaTech identified a potential SC Space vulnerability in the challenge program. The challenge program is not vulnerable due to random padding added at the end of the compressed file contents to prevent a viable SC Space vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.3.4.2.8 WithMi 3

A.3.4.2.8.1 Question 055 (AC Space, Intended Vulnerable, Answered Yes)

Response: GrammaTech responded that the *net.robotictip.compression.ZlibCompressor.unzip* method was vulnerable.

Why vulnerable code can be triggered by an attacker:

We can create a file containing slightly more than 120M of zeros. After compression into a Zip file, the archive is within the budget limit and can be send through the network within seconds. On the receiver side, the decompression initiated by net.robotictip.compression.ZlibCompressor.unzip creates a file on disk that is > 120M, i.e. the original size of the file.

How their *tools supported discovery of the vulnerability:*

Dual use of our tools and profiling help to find that the vulnerability is in net.robotictip.compression.ZlibCompressor.unzip and net.robotictip.compression.ZlibCompressor.unzipUtility. In particular, net.robotictip.compression.ZlibCompressor.unzip was identified as potential vulnerable code by our heuristics that rank methods by number of sinks.

Evaluation: The challenge program contains an AC Space vulnerability in its decompression implementation. The decompression algorithm uses run length encoding to compress and de-compress files sent. Compression is performed by identifying consecutive identical bits. The number of consecutive identical bits to be decompressed is an attacker controlled value. The check on this value starts at the attacker provided amount and counts down stopping when the value is equal to 0. An attacker can set this value to a negative number to exceed the resource usage limit. GrammaTech appears to have identified a vulnerability using only a consecutive number of 0's. This attack appears to take advantage of an unintended vulnerability in that the program may not check the maximum value variable which controls the number of consecutive bits represented on the remote side while processing inputs.

Post-Engagement Analysis Ruling: Correct

A.3.4.2.9 WithMi 4

A.3.4.2.9.1 Question 045 (AC Space, Intended Vulnerable, Answered Yes)

Response: GrammaTech responded that *this code is vulnerable to the same attack as withmi_3, Question 55.*

Why vulnerable code can be triggered by an attacker:

We can create a file containing slightly more than 120M of zeros. After compression into a Zip file, the archive is within the budget limit and can be send through the network within seconds. On the receiver side, profiling shows that com.digitalpoint.smashing.AwesomeSmashing.shrink takes a very large amount of time, and the full decompression of the file did not terminate after an overnight run. We stopped it while it was ~100M.

How their *tools supported discovery of the vulnerability:*

The method com.digitalpoint.smashing.AwesomeSmashing.shrink was marked as potential vulnerable code by our tools, as well as several methods related to files transfer and compression/decompression.

Evaluation: The challenge program contained an intended AC Space vulnerability. The challenge program implements run length encoding as part of its decompression scheme. This compresses consecutive identical bits within a file. The implementation in this challenge program contains a broken input guard which allows a user to bypass the maximum size enforced by the program. The number of bytes de-compressed is tracked using an integer therefore an attacker can cause this variable to overflow bypassing the input guard and triggering an AC Space vulnerability. The vulnerability identified by GrammaTech is not the intended vulnerability due to the fact that 120 million is less than the maximum value of an integer. This attack appears to take advantage of an unintended vulnerability in that the program may not check the maximum value variable which controls the number of consecutive bits represented on the remote side while processing inputs.

Post-Engagement Analysis Ruling: Correct

A.3.4.2.10 WithMi 5

A.3.4.2.10.1 Question 026 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: GrammaTech responded that they *believe there is an SCL for the same reason as for question 10 of withmi_1 (see justification 6). For this question, we also assume that the attacker is able to watch all packets in the network, including packets that are sent between users in the chat he is not invited to. The vulnerable method is transmitMessage in com.techtip.chatbox.Byper.java.*

Evaluation: Unlike in WithMi 1, this variant of the challenge program pads chat state messages to the same length; therefore, an attacker is unable to uniquely map the encrypted chat state message to the number of users in a chat.

Post-Engagement Analysis Ruling: Incorrect

A.3.5 Iowa State University

A.3.5.1 Iowa State Overview

Iowa State responded to 12 questions with a 33% accuracy. Iowa State answered 4 AC Space questions with 50% accuracy but had a 0% accuracy for all side channel questions.

Table A-51: Engagement 3 ISU Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	0	0	0	0
SC Time	1	1	100	0	0
SC Space/ Time	1	0	0	0	0
AC in Space	4	3	75	2	50
AC in Time	5	2	40	2	40
Total	**12**	**6**	**50**	**4**	**33**

Table A-52: Engagement 3 ISU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	9	4	**44**	2	**22**
Not Vulnerable	3	2	**67**	2	**67**
Yes Answer	5	4	**80**	2	**40**
No Answer	7	2	**29**	2	**29**

A.3.5.2 Iowa State Specific Responses

A.3.5.2.1 Airplan 1

A.3.5.2.1.1 Question 046 (AC Space, Intended Not Vulnerable, Answered No)

Response: Iowa responded that *the classes called XXXGuide were identified as the entry points to the application for handling user requests. After this we used the Loop Call Graph to identify reachable loops from each entry point. There are 13 methods in these entry point classes for handling user HTTP requests. There are 75 methods in the Loop Call Graph from these entry points, which consist of 61 loops in the methods (out of 270 loops in the application).*
We chose the following loops to investigate because they involved Graph traversal (invoked methods in Graph or iterated over Vertex/Edge)
Accomodation#accomodation
Accomodation#search
CrewCoordinator#composeCrewSchedulingGraph
CrewCoordinator#generateComplementaryFlightsMap
CrewCoordinator#pullCrewAssignments
ConnectedAlg#isConnected
OptimalTrail#calculateOptimalTrail
The loops in Accomodation methods "accomodation" and "search" were particularly interesting because the loop nesting depth was deep. This was identified using the Loop Call Graph and Loop Nesting Depth filter. These loops are also of interest because they are reachable from multiple of the program entry points. Using our Loop Catalog file and loop smart view scripts we investigated each of the loops involved in the Graph algorithm operations with non-trivial terminating conditions. We could not craft an input which would exceed the resource budget.

Evaluation: The application did not contain an intended AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.5.2.1.2 Question 052 (AC Time, Intended Vulnerable, Answered No)

Response: Iowa responded that t*he classes called XXXGuide were identified as the entry points to the application for handling user requests. After this we used the Loop Call Graph to identify reachable loops from each entry point. There are 13 methods in these entry point classes for handling user HTTP requests. There are 75 methods in the Loop Call Graph from these entry points, which consist of 61 loops in the methods (out of 270 loops in the application). We chose the following loops to investigate because they involved Graph traversal (invoked methods in Graph or iterated over Vertex/Edge)*
Accomodation#accomodation
Accomodation#search
CrewCoordinator#composeCrewSchedulingGraph
CrewCoordinator#generateComplementaryFlightsMap
CrewCoordinator#pullCrewAssignments
ConnectedAlg#isConnected
OptimalTrail#calculateOptimalTrail
The loops in Accomodation methods "accomodation" and "search" were particularly interesting because the loop nesting depth was deep. This was identified using the Loop Call Graph and Loop Nesting Depth filter. These loops are also of interest because they are reachable from multiple of the program entry points. Using our Loop Catalog file and loop smart view scripts we investigated each of the loops involved in the Graph algorithm operations with non-trivial terminating conditions. We could not craft an input which would exceed the resource budget.

Evaluation: The intended vulnerability in this application was in the shortest path determination. Given a route map, the application calculated the shortest path using A-star search with an inconsistent heuristic.

Post-Engagement Analysis Ruling: Incorrect

A.3.5.2.2 Airplan 2

A.3.5.2.2.1 Question 006 (SC Space, Intended Vulnerable, Answered No)

Response: Iowa responded that *the secret here is the number of airports. For a side channel in space to get the number of airports, we have to find a response, where only the size of the response is directly proportional to the number of airports and the additional data in the response (like flight information) is constant on all responses. Only then, we would be able to deduce the number of airports by a side channel in space The pullAirports() method in com.networkapex.airplan.prototype.RouteMap class pulls in all the airports. This was treated as a secret for our tool. The handleXXX methods in HttpExchange are the entry points that can be observed where data is sent over the network. Our tool (by creating IPCG's) helped us audit all cases and we could not find a case where a response is only based on number of airports. Therefore, we conclude that there is no side channel in space.*

Evaluation: When a user inputs a route map, the user is redirected to a page where a matrix representing the route map is displayed. The route map matrix is represented in a table template

which leaks the number of airports in the user's route map – encrypted data size is proportional to n^2 given a route map containing n airports.

Post-Engagement Analysis Ruling: Incorrect

A.3.5.2.3 BidPal 1

A.3.5.2.3.1 Question 014 (SC Time, Intended Vulnerable, Answered Yes)

Response: Iowa responded with:

Statement of how observables relate to the secret:
Observable - Time it takes to receive response from server once the request is issued.
Request issued - Bid comparisons. Users can make a bid and then compare the bid with other existing bids to check whether their bid is highest. This process involves validation of the bid by server. A side channel exists in this code which leaks information about private key of the highest bidder.

Side channel strength: Statement on why observables reveal secret even in the worst case:
Attacker is trying to get the private key of a user (he gets public key by connecting with him, another permitted operation. Thus he has complete information on the victim). It is a 2048 bit key. The SC leaks one bit at a time. So in best case he will do it in 2048 operations. Even if one allows for the noise in side channel (measurement errors in network traffic) he still gets to revise his guess about next bit about 73000 times (budget - 75000 operations). But as he starts guessing more and more bits correctly, the number of choices for the key will decrease exponentially and the probability of the attack succeeding is very high.

Statement on why the side channel can be triggered by the attacker:
Attacker creates a bot to act as a seller and creates an auction. Then he waits till some buyer bids on it. He then creates another bot who will bid on the same auction. When it tries to check whether the bid is highest or not (by making a request with message data type - bid comparison) he supply a different private key, making it an invalid bid. This will be done using the check in the loop described below which functions differently for different bits and leaks information. It also doesn't have the guards using Random API which mitigate the strength of the side channel by introducing the noise.

Evidence from our tooling:
1. Using Loop Reachability Filter and view, we identified 9 Loops reachable from the Bid value sent by a victim bidder:
2. The Loop Projected Control Graph (below) for the loop in the MgMultiplier.exponentiate method reveals a differential branch that depends on whether a bit in the secret key is 0 or 1 (the exponent controls the differential branch). On one path, the mgMultiply method is called twice and on another, it is called once.
3. We then compute a reverse data flow on the exponent and intersect it with the Loop Call Graph (LCG) to get all the methods which will reach this program point.
The leaf of the above graph is the method containing the differential branch, namely exponentiate. A close inspection by analyst reveals that the server handler can issue a request for bid comparison and it goes through the validation process. This can be done even when auction is running and thus seller has no control to prevent this unless he decides to end the

auction. There is also no limit on how many bids/bid comparisons one can do and attacker can keep on steadily guessing the private key of the highest bidder one bit at a time.

Evaluation: Iowa identified a potential SC Time vulnerability in the MgMultiplier.exponentiate() method. This method is used in the decryption of messages along with the comparison of bids. UMD referenced a research paper with an exploit against the RSA implementation in this challenge program. This represents an in-scope unintended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.5.2.4 Info Trader

A.3.5.2.4.1 Question 002 (AC Time, Intended Vulnerable, Answered Yes)

Response: Iowa responded that *a user can upload a series of documents where each document includes a hyperlink to every document before. After each document is posted to the server, the server updates the Sitemap.xml file by recursively traversing down each element's children. When the N^{th} document (starting with $N = 0$) is uploaded it's node becomes the root node in this graph of 2^N nodes. This traversal becomes very expensive because each nodes is written to the Sitemap.xml file.*
On the reference platform, after the 16th document is uploaded the application took ~1290 seconds to respond. The amount of budget used at this point is 2861 bytes.

Analysis:
We started by looking for the entry points to the program and identified the entry point DocumentParser.doPost. Of the 143 loops in the decompiled program we analyzed, we identified 92 loops loops reachable from the DocumentParser.doPost via in the Loop Call Graph smart view. We further analyzed interactions of loops and methods in the app with different subsystems using our Loop Filter and loop catalog. We identified 6 Methods in the app that interact with the Serialization Subsystem, which are of interest: These 6 methods contain four potentially vulnerable loops (vulnerable loop header is shaded dark; analysis on decompiled code):
We inspected these 4 loops using our Loop Termination Projected Control Graph (LPCG) smart view, and found that the loop in SiteMapGenerator.serializeNode() could cause an AC vulnerability for this question. The LPCG also shows a recursive call to serializeNode.
The above loop is reachable from the entry point via the call chain: DocumentParser.doPost() -> putDoc() -> SiteMapGenerate.getSiteMap() -> serializeTree() -> serializeNode()
We performed a dynamic analysis on the reference platform to confirm the AC vulnerability according to the question's budget and resource constraint. We have attached an exploit program (Question_002-Exploit) to demonstrate the vulnerability.

Evaluation: The challenge program does not contain an intended AC Time vulnerability. Northeastern identified an unintended vulnerability in the challenge program. Iowa identified the potential vulnerability in the document traversal algorithm similar to Utah. Iowa's exploit creates a complex tree structure in the document traversal. After 16 documents are loaded, the resource usage limit is exceeded; however, this requires 5353 bytes, which exceeds the input budget of 5000 bytes. Northeastern's exploit creates a long document chain with a loop at the end. This eventually triggers a *stackOverflowException* but not before the resource usage limit is exceeded.

Utah flagged the document traversal algorithm as vulnerable and while their exploit was not valid, Northeastern's was.

Post-Engagement Analysis Ruling: Correct

A.3.5.2.4.2 Question 004 (AC Space, Intended Vulnerable, Answered Yes)

Response: Iowa responded that *a user can upload a series of documents where each document includes a hyperlink to every document before itself. After each document is posted to the server, the server updates the Sitemap.xml file by recursively traversing down each element's children. When the N^{th} document (starting with $N = 0$) is uploaded it's node becomes the root node in this graph of 2^N nodes. Each of these nodes is then written to the Sitemap.xml file.*

The given budget of 5,000 bytes allows the upload of 21 documents (the 0th document followed by 20 linking documents). These creates a graph with 2^20 (1,048,576) nodes. The application's initial state has a Sitemap.xml file of 1,351,371 bytes. With a target resource usage of 100,000 kb (102,400,000 bytes) we need to create 101,048,629 bytes of data storage. Therefore each node needs to create at least 93.37 bytes of data storage. By running our exploit program for Question 002 we were able to extract the Sitemap.xml file after each document is uploaded and compare the growth of the file. We found that the number of bytes created for each node in the graph continues to increase as documents are uploaded (due to the indentation whitespace and nested XML tags). After the 15th linking document is uploaded, each node in the graph creates 326 bytes of data storage in the Sitemap.xml file. This is well above the necessary target to reach the target of 100,000 kb by the 20th linking document. This ratio would cause the file to exceed the target after the 19th linking document is uploaded.

Due to the AC Time vulnerability in this program we ran out of time to upload the necessary documents to prove the AC Space vulnerability. See the Excel spreadsheet "info_trader_Q4_data.xlsx" for the numbers and calculations used for this proof. The graph below shows the size of Sitemap.xml exceeding the target resource usage.

Evaluation: The challenge program contained a broken input guard which while intended to prevent cycling allowed users to upload a document with the same name as the root node. If an attacker uploads a document linking to the document with the same name as the root, the resulting document traversals that generate the sitemap xml file will not cause a *StackOverflowError* but will result in a sitemap xml file which exceeds the resource usage limit. The observed resource usage limit of Iowa state's exploit was 6,496 kB which is less than the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.3.5.2.5 Linear Algebra Platform

A.3.5.2.5.1 Question 012 (AC Time, Intended Vulnerable, Answered No)

Response: Iowa responded that *the app has 4 possible operations:*
1) Matrix Multiply
2) Laplacian
3) Shortest Path
4) MST
As the standard algorithms for these operations are themselves quite expensive (ranging from

quadratic to cubic), we looked for a differential branch. In other words, we hypothesize that for an ACV in time to exist, there should be a branch that governs two paths: one that depends on some property of input and it does an inordinate amount of computation on that particular path, and another that follows the standard algorithms on rest of paths.
We first compute the loops reachable from the user input (reference to the json object containing matrix/graph) using our Loop Reachability smart view tool. This narrowed our search from 103 loops in the app to 56 loops.

Investigation of potentially vulnerable structures:

Case 1: Matrix Operations
Since this app accepts matrices in form of arrays, we decided to investigate loops which iterate over arrays first. These loops correspond to matrix multiplication and laplacian computation. This is done using our Loop Pattern Filter. This gets us to the 16 loops from the 56 loops:
After inspecting these loops we inspected the matrix multiplication and laplacian functionalities of the app. Matrix Multiplication is computed parallely by creating partitions (linear arrays). Then they are multiplied and copied into result matrix. Finally the threads are joined. This is expected behavior of a parallel matrix multiplication algorithm. Since this is the only way to compute matrix multiplication in this app, we rule out this possibility. Laplacian is also computed as expected by subtracting adjacency matrix from degree matrix. Again this is the only way this app computes Laplacians. So we rule out this possibility as well.

Case 2: Graph Operations
A quick look at our generate Loop Catalog for this app reveals the existence of loops using Iterables. Further inspection shows a class EdgeWeightedDirectedGraph whose vertices and edges are stored as Iterable Collections. This means that the shortest path and MST computation must involve these loops and they must be analyzed. We apply our Loop Pattern filter on the remaining 40 loops. This gets us to following 24 loops. Inspecting these helped us understand the remaining two functionalities. Shortest Path is computed using Dijkstra's Algorithm. The implementation appears to be standard Dijkstra. Standard Dijkstra's weakness is that it can't be used on graphs with negative weights. We looked for a guard against it and found one in the DijkstraSP method. It throws an exception if it found any negative weights. This is the only way this app computes SP. So we rule out this possibility. MST is computed using Prim's algorithm. Again the implementation seems to follow the standard algorithm (LPCG reveals the structure of loops similar to the ones in dijkstra, which is not surprising). This is the only way this app computes MST. So we rule out this possibility. As all four functionalities are performing as expected and there is no differential branch creating singularities in the app. Thus, we conclude that there is no algorithmic complexity vulnerability in time.

Evaluation: The challenge program contains an intended vulnerability in the parallelized matrix multiplication functionality. This vulnerability was in the balancing of tasks for the performance of the matrix multiplication. The balancing algorithm uses a metric based on the mean row skew of the first matrix submitted. Fast case: $|\mu_{row\ skew}| < \ln(2)$; slow case: $|\mu_{row\ skew}| > \ln(2)$. Iowa correctly recognized that the matrix multiplication function is parallelized but did not identify that the distribution of tasks between the two threads is user controlled. However, Iowa's response does provide some insight into their approach for evaluating vulnerabilities. From this response it is clear that they methodically investigated each of the challenge programs

capabilities looking in areas where traditional implementations of these algorithms are vulnerable.

Post-Engagement Analysis Ruling: Incorrect

A.3.5.2.6 Malware Analyzer

A.3.5.2.6.1 Question 031 (AC Time, Intended Not Vulnerable, Answered Yes)

Response: Iowa responded that: *our analysis revealed a potentially expensive operation in the Analysis.compute_cosine_similarity_helper which is called by both the compute_cosine_similarity(Sample s1, Sample s2) and the compute_cosine_similarity(Dasm s1, Dasm s2) methods. The recursion levels are bounded by a steps variable. Each caller supplies a hardcoded number of steps for the recursion depth (255 for samples and 664 for dasms). The numbers steps corresponds to the maximum length of the feature vector for each type. If there is a dasm sample with a maximum sized feature vector that is compared against a dasm sample with a smaller feature then each element of the feature vectors will be compared regardless. The dasm's feature vector is computed in generate_mnemonic_feature_vector. The maximum feature vector size can be achieved by maximizing the number of mnemonic Instructions as defined by the X86 class. The "add_dasm" operation can be used to add new dasm programs (detailed analysis of this code region can be found in Question 35). The "query_dasm" operation is used to compute the cosine similarity of all dasm programs and return the top 5 best matches. Since this operation compares the selected dasm sample against all other dasm samples, a single dasm with the maximum feature vector can be created with the "add_dasm operation" and then several very small dasms can be created and then one more "query_dasm" operations could be provided to invoke the expensive cosine similarity analysis logic.*
Use of Tooling:
We used the Loop Call Graph (LCG) to get all the methods containing loops reachable from the Server.serve() method, which is the entry point of this app (shown below).
Note: Additional analysis was done in Question 35 which contributed to this answer.

Evaluation: The challenge program does not contain an intended AC Time vulnerability. The exploit provided by Iowa exceeded the input budget by 2292 bytes and did not exceed the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.3.5.2.6.2 Question 035 (AC Space, Intended Vulnerable, Answered No)

Response: Iowa responded that *the input budget is 3000 bytes. The outfile size requirement is 80KB. The only file that is written to the disc (as identified by our IO_SUBSYSTEM interaction analysis) is written via the Logger.write_log_entry method. The only reference that is passed to the write_log_entry (as determined by Atlas dataflow analysis) is the Server.response global variable. The points-to analysis indicated that the Server.response variable is not aliased and only direct assignments to Server.response are used to set the value. Using Atlas data flow, the assignments to the "response" string are made in several locations that are all restricted to the Server class. Of these locations, there are 8 assignments made within loops and 5 assignments outside of loops (all assignments are within the Server.serve method and all assignments concatenate results to the existing string value). The analysis queries are shown below.*
First a selection of the global variable was made.

var response = selected Selects a set of all program graph elements contained within loop bodies

var loopBodies = org.rulersoftware.loop.util.LoopUtils.getControlFlowGraph(universe.nodesTaggedWithAny("LOOP_HEADER")).contained() Select all assignments to Server.response made within loops (the results are all in the Server.serve method).

var assignmentsInLoops = edges(XCSG.DataFlow_Edge).forward(response).intersection(loopBodies).nodesTaggedWithAny(XCSG.Assignment)Selects assignments outside the loop.

var assignmentsOutsideLoops = edges(XCSG.DataFlow_Edge).forward(response).nodesTaggedWithAny(XCSG.Assignment).difference(assignmentsInLoops)

Of the 5 assignments outside loops, 2 are concatenations of fixed lengths. The two variable length assignments outside of loops are worth considering (Cases 1A, 2A and 3A).

Case 1A:
In the case of an "add" operation there is a possibility to write a stack trace to the log file if an exception is thrown during an add that is not NumberFormatException or an IllegalArgumentException. A java.lang.StackOverflowError would generate a large stack trace. The stack overflows at roughly 1000 recursions deep and the long package names of "com.ainfosec.MalwareAnalyzer" plus the class name would put us just within range of the 80KB budget. Within the corresponding try block, there are only 3 callsites (Database.see, new Sample, and Database.add_sample). The forward call graph of each must contain some recursive calls in order to overflow the stack. Of the three callsites, the Sample constructor does actually call three recursive methods (Analysis.compute_average_byte_helper, Analysis.compute_byte_entropy_helper, and Analysis.count_bytes_helper), however each method contains a recursion guard that limits the recursion to a maximum of 255 levels deep. All three of these methods contain a wrapper method that initializes the steps to zero (so entering a negative value is not possible) and none of the helper methods are called directly (so there is no way to control the levels of recursion). This rules out the possibility of a StackOverflowError and case 1A can be safely ignored.

Case 2A:
Also in the "add" operation, is the catch block for the IllegalArgumentException type. In this block the toString of the exception is called which accesses the message set in the IllegalArgumentException constructor. Using Atlas to search for inputs to the IllegalArgumentException constructor shows that are 3 locations in the new Sample constructor where a custom message is passed, however the message is a substring of the String the new Sample was constructed with, which means it is bounded in length by our input budget. So case 2A can be safely ignored.

Case 3A:
In the "query" operation, the value of the "feature_vector_output" local variable is concatenated to the response. The feature_vector_output variable is a string that is concatenated

in a loop, that is bounded by at most 5 iterations. Each iteration of concatenations adds the length of the md5, and the feature vector (an array of 255 integers), which is nowhere near enough to exceed our budget so case 3A can be ignored. Of the assignments inside the the bounds of each loop should be considered (Cases 1B, 2B, 3B, 4B, and 5B).

Case 1B:
In the "query_dasms" operation, the response is concatenated in a loop at most 5 times. The contents of the strings are limited in the same way as case 3A and can be safely ignored.

Case 2B:
In the "get_function_entrypoints" operation there is a loop that is bounded by the number of control flow graphs. The output is a concatenation of hex integers of each CFG entry point. Through an analysis for case 5b (described below) a call dasm instruction is required to create a call entry point. The smallest call dasm instruction is of the form: "<address>: 00 call 0x<target>", where <address> is a minimum of two characters and <address> is at least one character. Combined with the overhead of the required get_function_entrypoints options, we can at most create 175 call instructions with the available input budget. This bounds the loop for case 2B at 175 iterations which is not enough to exceed the file size budget.

Case 3B:
In the "list_dasms" operation is bounded by the number of md5 hashes in the database. There is a way to enter an md5 hash without entering the required feature vector by entering an invalid "add" request. The new md5 is added with Database.see and then if an error occurs (in the case that the next two input lines are invalid) then the operation is aborted, but the md5 is not removed from the database. However this cannot be used to exceed the output budget because we would have to create at least 2500 md5 entries (each requiring 3 + 32 + 2 bytes of input) to reach an output of 80KB. This exceeds our attack budget and so case 3B can be ignored.

Case 4B:
In the "query" operation the response is concatenated in a loop for up 5 iterations (corresponding to the 5 most similar feature vector samples). The concatenation is very similar to case 3A and cannot exceed the budget).

Case 5B:
The "get_cfg" operation is the most promising of the concatenations to the response variable in a loop. The iteration is bounded by the number of control flow blocks in a given entry point. For each iteration the instruction address, operand, and its successors are printed. The successors a printed in an inner loop. The "add_dasm" operation allows the user some control of the upper bound (within the input budget) of the number of control flow graphs. To add a new set of dasm instructions the "add_dasm" command, followed by a new md5 hash, and at least 3 lines of dasm instructions must be sent. Atlas and taint analysis was used to follow the control and data dependencies through the parsing logic in DasmHelpers.build_dasm.

The DasmHelpers.parse_instructions method filters input lines to lines that match the regular expression "\\s(?<address>[0-9a-fA-F]+):\\s+(?:[0-9a-fA-F]{2})+\\s*(?<instr>\\S.*)".*

The smallest input for this regular expression is of the form "<address in hex>: <2 digits hex> <dasm instruction> 0x<1 digit hex>".
Next the DasmHelpers.discard_badly_decoded_instructions method discards instructions that match either "(bad)" or simply "[0-9]+".

Next function entry points are gathered in the DasmHelpers.get_function_entrypoints method. A function entry point is defined by a dasm call instruction that matches "call 0x(?<targetaddr>[0-9a-zA-Z]+)".

Finally the CFG for each function is created in the DasmHelpers.make_function_cfgs method and DasmHelpers.make_function_cfg method.
The make_function_cfg body contains a worklist style algorithm (and can be found by the RULER Filter View after filtering monotonic loops) that depends on the key_queue list. The key queue is a set of dasm instruction addresses. Every time a new instruction is read or referenced (ex: a jump at address 0x01 to address 0x04 is read at 0x01 and references instruction at 0x04) the address is added the worklist (if it hasn't been seen before). A Project Control Graph (PCG) of the add events shows that there are several events in which new addresses can be added to the work list (PCG where key_queue.add(...) is selected as the events) is shown below.

Callsites can be queried programmatically with the following Atlas query.methods("make_function_cfg").contained().nodesTaggedWithAny(XCSG.CallSite).selectNode(XCSG.name, "add(...)"). Unfortunately, there is no way to game this worklist to grow indefinitely by say adding two jump instructions that point two each other because a separate set of "seen" instructions is maintained and a PCG of add(...) events for the worklist and add events for seen instructions reveals that there is no path that escapes this check.

Since the response also includes the successors of an instruction we should create a PCG for the successor add events (show below). There are 3 cases where successors are added: unconditional jumps, conditional jumps, and function calls. The PCG reveals that unconditional jumps allows 2 opportunities to add successors. With this knowledge we can begin building an input to maximize the CFG blocks and successors. Since the "get_cfg" operation is with respect to only a single entry point, we should create a single call instruction that contains one large CFG. So our input should begin with:
add_dasm
8aed92dfebabc84e10a8d20983bea666
pei-i386
0: 00 call 0x0
The unconditional jumps get the most bang for the buck by allowing two successors. Following the X86.is_a_conditional_jump method tells us that a "ja" instruction is the shortest named conditional jump instruction. Since a block is actually created for every referenced instruction and there is no check for whether a referenced instruction actually exists we can create a second basic block by referencing an instruction that does not exist. For example the next lines of our exploit would be:
1: 00 ja 0xb9
2: 00 ja 0xba
3: 00 ja 0xbb
…

With this pattern we can create 188 conditional jump instructions. Starting our jump targets at 185 lets us maximize the number of short instruction addresses that do not exist. However even with all of these maximizations in place the output of the get_cfg for the corresponding add_dasm is only 12KB. Test code for the relevant extract segments of this case have been included. Note that the referencing invalid addresses only added ~4KB extra. With this in mind this case can be safely ignored as well.

Evaluation: The challenge program contained an intended AC Space vulnerability in the packer detector functionality. Given a 256-integer feature vector, while calculating the average value heuristic an attacker can bypass the input guard to trigger an arithmetic exception. This exception is written to the log file. Normal log writes are overwritten, but the exception causes the entire stack trace to be appended to the log file. Iowa state observed that it is possible to *write a stack trace to the log file it an exception is thrown during an add that is not NumberFormatException or an IllegalArgumentException*. They investigated the potential for a *StackOverflowError* as well. Iowa state missed the possibility for an *ArithmeticException* caused by a division by 0. The detail of Iowa state's response provided insight into their investigation of the vulnerability; however, given that this is a live engagement, a response of this length while appreciated was not required.

Post-Engagement Analysis Ruling: Incorrect

A.3.5.2.7 SmartMail

A.3.5.2.7.1 Question 039 (SC Time/Space, Intended Vulnerable, Answered No)

Response: Iowa responded that their *audit shows that there is no way to get a list of all email addresses stored on the app. The only way to get an email address would be to guess the email address. We did discover couple of space based side channels based on the secret (admin email). Given an email address would say whether that email address is a admin(secure) email address or not.*

1. If an email is sent to an admin(secure) email address, then the log file addlog1.log gets incremented in size. The taint from write to the logs shows that secure emails are logged to addlog1 and others are logged to addlog2.log

2. If you send an email to a secure email address and then try to get the email back for that mailbox, then you don't see the email you just sent, but just the default email. This tells you that this email address is a secure(admin) email address.

Given the size of email address (no including the domain name) is restricted to 25 characters, and each could be an alphanumeric character, it is not possible to guess in the given budget. We are evaluating whether there is some segmented side channel attack that can give us the answer in the budget. Since our tools have not found evidence of it yet, we are giving an official answer of No at this time.

Evaluation: The challenge program contained an SC Time/Space vulnerability which leaked the administrator of a given mailing list. The vulnerability was in the log file writes that occur when a message is sent. The set of all recipients of a given message (which includes the mailing list administrator and each of the subscribers) is alphabetized and written to a log file with the mailing list subscribers written to one log file and the administrator written to another. Using this

side channel an attacker can run a binary search for each character and determine each subsequent character of the administrator's email beginning at the first character. Iowa identified that there were two different log files written to when an email is sent. They recognized that one applies to the administrator and the other is used for other users. Iowa did not however recognize that the sequence of log file writes leaks the administrators email, and that the order in which emails are written to the logfile is deterministic – alphabetical.

Post-Engagement Analysis Ruling: Incorrect

A.3.5.2.8 Tour Planner

A.3.5.2.8.1 Question 008 (AC Time, Intended Not Vulnerable, Answered No)

Response: Iowa responded that o*f the 419 loops in the app, we were able to select 13 loops that are reachable from the HTTPRequest parameter passed to the TourServlet. We hypothesized that the AC vulnerability, if exists, should be rooted in one of these 13 because TourServlet is the entry point when the client requests a tour calculation, and the parameter contains the list of places to compute the tour over.*
We audited the 13 loops and did not find any loop that could potentially exceed the resource constraint of 30 seconds using only 500 bytes of request size. The only suspicious depthFirstWalk has a recursive call but that is expected. Therefore we concluded that the app is not vulnerable to AC vulnerability for this question.

Evaluation: The challenge program does not contain an intended AC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.5.2.9 Tweeter

A.3.5.2.9.1 Question 017 (AC Space, Intended Vulnerable, Answered Yes)

Response: Iowa responded that a *tweet can be made with a post to /tweet and is received by TweetingController.tweet method.*
After authenticating the user the tweet input is passed to the SpellingCorrectionService.correctAndSaveTweet method. The correctAndSaveTweet method creates a background task to correct the spelling of the tweet. There are no bounds on the size of the tweet at this point, so our input could be the maximum size of our allowed budget. There is no working UI implementation for making a tweet (the tweet button does nothing). You can post to the back to endpoint, but you have to make a form and fill in the correct _csrf token. Modifying the DOM with the following and editing the CRSF will make the post to the proper endpoint.
<form action="/tweet" method="post">
*Tweet:
*
*<input type="text" name="tweet">
*
<input name="_csrf" value="9a4bcd2e-4dea-4bb5-bb59-6ef3210ffb3d" type="hidden">
<input type="submit" value="Submit">
</form>
The maximum length of a word in a tweet is the maximum length of the word in the dictionary as calculated by Spelling.loadDictionary (the longest word in the dictionary). Quickly browsing the dictionary reveals "resourcefulness" as a candidate for longest word. An input consisting of the "resourcefulness" word with the typo in the last character position (e.g. resourcefulnesd") causes a high number of string operations because the edit distance window size is only 2

characters wide. Providing an input of "resourcefulnesd " concatenated repeatedly exceeds the memory budget for this application. A PoC exploit and the relevant code snippets are included in the TwitterPoC project.
This vulnerability was found by searching for deeply nested loops, filtering by loop monotonicity, and tainting with user controlled inputs. The Loop Call Graph for the methods of interest is shown below.

Evaluation: Iowa identified the intended vulnerability within the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.6 Northeastern University

A.3.6.1 Northeastern Overview

Northeastern answered the most questions in engagement 3 (51) and had a 67% Accuracy.

Table A-53: Engagement 3 NEU Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	7	78	6	67
SC Time	11	7	64	4	36
SC Space/ Time	1	0	0	0	0
AC in Space	15	11	73	11	73
AC in Time	15	13	87	13	87
Total	**51**	**38**	**75**	**34**	**67**

Table A-54: Engagement 3 NEU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	34	26	**76**	22	**65**
Not Vulnerable	17	12	**71**	12	**71**
Yes Answer	31	26	**84**	22	**71**
No Answer	20	12	**60**	12	**60**

A.3.6.2 Northeastern Specific Responses

A.3.6.2.1 Airplan 1

A.3.6.2.1.1 Question 046 (AC Space, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that t*he tool reported a number of attacker-influenced loops that allocate space resources. However, none of the loops seem to allocate enough memory to exceed the allotted amount or allow an attacker to cause lots of memory to be allocated. Dynamic testing was used to confirm in cases where we weren't sure.*

Evaluation: The challenge program did not contain an intended AC Space vulnerability

Post-Engagement Analysis Ruling: Correct

A.3.6.2.1.2 Question 052 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that they f*ound a complexity attack in TextFileLoader, the number of airports is not checked until after loading, so an attacker can provide a file with 3800 airports. For each airport it will run grabAirport() which has an expensive loop over all the previous airports and will exceed the allotted time. The loop was one of the list flagged as potentially expensive given its dependence on attacker input and control flow structure.*

Evaluation: The intended vulnerability in this application was in the shortest path determination. Given a route map, the application calculated the shortest path using A-star search with an inconsistent heuristic. Northeastern identified an unintended vulnerability in the challenge program. The AC team confirmed this unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.2 Airplan 2

A.3.6.2.2.1 Question 006 (SC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that the *response of* [a] *matrix request is* $O(n^2)$ *bytes in number of airports, so an attacker that observes this encrypted message can infer the number of airports reliably. The tool flagged the influence of secret data on the construction of the matrix and its transmission on a network channel*.

Evaluation: Northeastern identified the intended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.2.2 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that they *found a complexity attack in TextFileLoader, the number of airports is not checked until after loading, so an attacker can provide a file with 3800 airports. For each airport it will run pullAirport() which has an expensive loop over all the previous airports and will exceed the allotted time. The loop was one of the list flagged as potentially expensive given its dependence on attacker input and control flow structure*.

Evaluation: The intended vulnerability in the challenge program was in the sort method. Provided a list of n items where n is divisible by 6, the implementation of merge sort performs many unnecessary merges resulting in an AC Time vulnerability. Northeastern identified an unintended vulnerability in the challenge program. The AC team confirmed this unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.3 Airplan 3

A.3.6.2.3.1 Question 013 (SC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *Map_properties response has different sizes based on whether or not it's fully connected. Not fully connected is larger. So an attacker just needs to observe this encrypted message size. The tool flagged the influence of the connectedness property on message construction*.

Evaluation: Northeastern identified the intended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.3.2 Question 022 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: Northeastern responded that r*esponse of matrix request is $O(n^2)$ bytes in number of airports, so an attacker that observes this encrypted message can infer the number of airports reliably. The tool flagged the influence of secret data on the construction of the matrix and its transmission on a network channel*.

Evaluation: The challenge program displays the graph submitted by a user in a standard form which does not reveal the size of the graph. The graph is displayed in table form with entries padded with random whitespace; therefore, the number of airports in a user's route map cannot be uniquely mapped to the size of a given packet.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.3.3 Question 025 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that they *found a complexity attack in TextFileLoader, the number of airports is not checked until after loading, so an attacker can provide a file with 3800 airports. For each airport it will run pullAirport() which has an expensive loop over all the previous airports and will exceed the allotted time. The loop was one of the list flagged as potentially expensive given its dependence on attacker input and control flow structure*.

Evaluation: The challenge program contained an intended vulnerability in calculating the flight crew required for a given route map. This is done by solving the max flow problem using the Ford Fulkerson algorithm. Northeastern identified an unintended vulnerability in the challenge program. The AC team confirmed this unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.4 Airplan 4

A.3.6.2.4.1 Question 009 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that they *found a complexity attack in TextFileLoader, the number of airports is not checked until after loading, so an attacker can provide a file with 3800 airports. For each airport it will run pullAirport() which has an expensive loop over all the previous airports and will exceed the allotted time. The loop was one of the list flagged as potentially expensive given its dependence on attacker input and control flow structure*.

Evaluation: The challenge program contained an AC Time vulnerability in the shortest path algorithm. The challenge program uses an implementation of Dijkstra's algorithm which allows negative costs. An input route map with negative cost values can be used to trigger an AC Time vulnerability. Northeastern identified an unintended vulnerability in the challenge program. The AC team confirmed this unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.4.2 Question 030 (SC Space, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that their *tool reported the influence of secret data on attacker observables. However, the properties page has the same size whether or not it is fully connected so there is no side channel in space that can allow an attacker to determine if the graph is fully connected*.

Evaluation: Northeastern correctly responded that the application did not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.4.3 Question 048 (AC Space, Intended Vulnerable, Answered No)

Response: Northeastern responded that their *tool reported a number of loops influenced by attacker input. However, we couldn't find any loops where the user input could cause enough memory usage such that the program would exceed the memory limit*.

Evaluation: The challenge program contained an AC Space vulnerability. The program allows users to input route maps in an xml format. The XMLFileLoader class allows inputs which reference pre-defined airports resulting in an xml bomb. Northeastern did not identify the intended AC Space vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.5 BidPal 1

A.3.6.2.5.1 Question 014 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: Northeastern responded that *MgMultiplier.exponentiate() has a timing difference between paths that are controlled by individual bits of D. However,* I *do not have enough time to PoC the attack*.

Evaluation: The implementation of RSA in this challenge application does not contain an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.5.2 Question 044 (AC Time, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that *due to time constraints,* they *only ran automated tools on this challenge (control flow graph recovery, recursion finding and loop analysis). We did not find any non-terminable loop or recursive function calls. Hence we made a reasonable guess, which is there is no side channel in time.*

Evaluation: The challenge question asked if the challenge program contained and AC Time vulnerability. Northeastern responded that there was *no side channel in time* in the challenge program. Judging by the earlier part of the response it appears they may have meant there is no AC Time vulnerability in the challenge program. The application did not contain an intended AC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.6 BidPal 2

A.3.6.2.6.1 Question 037 (SC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *MgMultiplier.exponentiate() has a timing difference between paths that are controlled by individual bits of D. However, I do not have enough time to PoC the attack*.

Evaluation: The challenge program contained an SC Time vulnerability which leaked a user's bid. The time to create a bid comparison message is proportional to a function of the user's bid. The side channel identified by Northeastern does not appear to leak this information.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.6.2 Question 041 (AC Time, Intended Vulnerable, Answered No)

Response: Northeastern responded that *due to time constraints,* they *only ran automated tools on this challenge (control flow graph recovery, recursion finding and loop analysis). We did not find any non-terminable loop or recursive function calls. Hence we made a reasonable guess, which is there is no side channel in time.*

Evaluation: The challenge question asked if the challenge program contained and AC Time vulnerability. Northeastern responded that there was *no side channel in time* in the challenge program. Judging by the earlier part of the response it appears they may have meant there is no AC Time vulnerability in the challenge program. The application contained an AC Time vulnerability. As part of the peer-to-peer auction protocol, after a comparison message is generated, checks are performed for the validity of the username and auction ID. Given an attacker with a username containing "NO_USER" (variable defaultUser) and an auction ID containing "NO_AUCTION_ID" (variable defaultAuctionID) the AuctionProcessor.checkComparisonBytes calculates a fake checksum which fails and causes the message to be re-created.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.7 Collab

A.3.6.2.7.1 Question 018 (AC Time, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that *the tree used to search for events is kept balanced. We checked all the operations and all the identified loops and recursive calls. We think that all of them terminate within the allowed time.*

Evaluation: The challenge program did not contain an intended AC Time vulnerability

Post-Engagement Analysis Ruling: Correct

A.3.6.2.7.2 Question 028 (SC Space, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that their *tool enumerated the points at which secret information is transmitted by the application. The application communicates using fixed-length fields and structures. Additionally, the number of packets returned by most commands is fixed. Finally, the commands returning a variable number of packets (e.g., SEARCH), do not reveal any secret information. Therefore, we think that there is no space side-channel vulnerability in this application.*

Evaluation: The challenge program did not contain an intended SC Space vulnerability

Post-Engagement Analysis Ruling: Correct

A.3.6.2.7.3 Question 042 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: Northeastern responded that their *tool reported that untrusted input flows to methods that manipulate a B-Tree structure. This program queries/updates this B-Tree structure every time events are searched/added. The time needed to update the B-Tree is not constant since, sometimes, the function "split" is called (when the bucket of values associated with a node of the tree gets full). An attacker can measure the time needed to add an event to understand when a split operation in the tree happened. Empirically, we noticed that the insertion time is sometimes longer and we verified (by using a debugger) that the cause of this slowdown is indeed the*

execution of the function: collab.SchedulingSandbox.split(DataNode orignode). An attacker can also query (using the "search" operation) all the events it is suppose to see and compute when a specific insertion is suppose to trigger a split operation. If the attacker's prediction is not correct, this means that an extra element is present in the tree in the node modified by the last insertion. This extra element must necessary be an audit event. Therefore, an attacker can progressively narrow-down the location in the tree where an audit event is stored. In turn, this allows an attacker to logarithmically (within the allowed input budget) narrow-down the range of values the audit event has. Finally, an attacker can query a hypothesized value for an audit event by trying to insert an event with the same id and check whether the error: "DuplicateEvent" is returned.

Evaluation: This challenge program did not contain an intended SC Time vulnerability. The *split(DataNode orignode)* method reference by Northeastern performed 3 log writes and part of its implementation resulting in an SC Time vulnerability. This side channel was non-locally balanced in the SchedulingSandbox.add() method resulting in a balanced insert operation.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.8 Info Trader

A.3.6.2.8.1 Question 002 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that their *tool reported a recursive method invocation that it was not able to establish a base case for.*
Given the way in which the program processes links, a cycle in the hyperlink graph will lead to infinite recursion.
This happens in the following line:
infotrader.dataprocessing.SiteMapGenerator.serializeNode(SiteMapGenerator.java:404)
A simple cycle (e.g., d1 links to d2 and d2 links to d1) generates a java.lang.StackOverflowError due to recursion before the 1000 second limit. However it is possible to create a structure of links and documents exceeding this limit when processed.
Specifically, the following input triggers the vulnerability:
./postdocument.sh d20.txt; ./postdocument.sh d19.txt; ./postdocument.sh d18.txt; ./postdocument.sh d17.txt; ./postdocument.sh d16.txt; ./postdocument.sh d15.txt; ./postdocument.sh d14.txt; ./postdocument.sh d13.txt; ./postdocument.sh d12.txt; ./postdocument.sh d11.txt; ./postdocument.sh d10.txt; ./postdocument.sh d9.txt; ./postdocument.sh d8.txt; ./postdocument.sh d7.txt; ./postdocument.sh d6.txt; ./postdocument.sh d5.txt; ./postdocument.sh d4.txt; ./postdocument.sh d3.txt; ./postdocument.sh d2_1.txt ; ./postdocument.sh d2_2.txt ; ./postdocument.sh d1.txt
Where d20.txt contains a document named d20, d19.txt contains a document named d19 linking to d20, d18.txt contains a document named d18 linking to d19, and so on up to d2_1.txt.
d2_1.txt is named d2 and links to a document named d1.
d2_2.txt is named d2 and links to the document d3 AND d1.
d1.txt is named d1 and links to d2

Evaluation: The challenge program did not contain an intended AC Time vulnerability. Northeastern appears to have identified an unintended AC Time vulnerability in the challenge program. Northeastern's exploit is a reference chain of 17 document with a loop at the end requiring a total of 20 documents. The loop at the end of the document chain has been confirmed to trigger a *java.lang.StackOverflowError*. Once this occurs, the server returns an error to the

user. Northeastern attempts to utilize this by creating a document chain long enough to prolong the *java.lang.StackOverflowError* such that the resource usage limit of 1000 seconds is exceeded. Northeastern has identified an unintended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.8.2 Question 004 (AC Space, Intended Vulnerable, Answered No)

Response: Northeastern responded that *given how documents are processed, the only file that could significantly increase its size is the sitemap xml file (which was among those reported as potentially untrusted input by the tool). However, the content of this file only contains part of the document and a non-recursive list of links. In fact, if the link structure does not contain a cycle, it is impossible to hit the 100,000 KB threshold. (also because only the first link is followed when the tree is serialized). On the contrary, if the link structure contains a cycle, then the execution will not terminate (or it will terminate with a StackOverflowError) and, therefore, no sitemap file will be generated.*

Evaluation: The challenge program contained a broken input guard which while intended to prevent cycling allowed users to upload a document with the same name as the root node. If an attacker uploads a document linking to the document with the same name as the root, the resulting document traversals that generate the sitemap xml file will not cause a StackOverflowError but will result in a sitemap xml file which exceeds the resource usage limit.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.9 Linear Algebra Platform

A.3.6.2.9.1 Question 005 (AC Space, Intended Vulnerable, Answered No)

Response: Northeastern responded that their *tool reported the influence of untrusted input on each operation. Each functionality was tested with increasingly larger inputs and the space complexity was not large enough to exceed limits given the constraints on the input.*

Evaluation: The challenge program does not contain an intended AC Space vulnerability. GrammaTech and Utah identified an unintended in-scope vulnerability within the challenge program. Linear Algebra Platform uses the MatrixSerializer.readMatrixFromCSV() method to parse input matrices. This method has two enforcement parameters enforceWidth and enforceSize which are used all but one of the possible input matrices. The shortest path operation of the matrix takes two input matrices, an adjacency matrix and a targetNodes vector representing the set of nodes to find the shortest path between. The vector of nodes is parsed with the enforceWidth and enforceSize parameters set to false. The readMatrixFromCSV method proceeds to allocate memory for the targetNodes vector using the user submitted arguments which results in a memory usage of 3,653.928 MB exceeding the resource usage limit of 2,500 MB. An illegal argument exception is then triggered once the method finds that the number of rows in the input matrix and the number of rows declared by the user do not match; however, this is after the memory has been allocated for the targetNodes vector.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.9.2 Question 012 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that their *tool reported the influence of untrusted input on the matrix muitiplication request. The parallelized matrix multiplication functionality has n^3 complexity, combine with using small floating point numbers (denormalized) ie*

0000000019088e-40 and 650x650 matrices and it will take more than the allotted time. (Took 4min5sec for one test).

Evaluation: The challenge program contains an intended vulnerability in the parallelized matrix multiplication functionality. This vulnerability was in the balancing of tasks for the performance of the matrix multiplication. The balancing algorithm uses a metric based on the mean row skew of the first matrix submitted. Fast case: $|\mu_{row\ skew}| = 0$; slow case: $|\mu_{row\ skew}| \geq \ln(2)$. Northeastern's exploit requires an input matrix where every value is the same. This results in a $\mu_{row\ skew}$ of 0 and should trigger the fast case of the operation. In the fast case, the number of operations performed is evenly divided among the two threads. Northeastern's exploit results in this case and therefore does not exercise the intended vulnerability in the challenge program. With the exploit described by Northeastern, we were able to exceed the resource usage limit of 225 seconds (251.421 seconds). Northeastern's exploit of using small floating point numbers exceeded the resource usage limit and represents an unintended vulnerability in the challenge program. The challenge program has a requirement for a minimum of 17 digits of precision for each input double. Northeastern recognized that there was no check on a minimum input greater than 0 and used this to exceed the resource usage limit. The AC team confirmed that this out-of-scope AC Time vulnerability is due to the *double* to *String* and *String* to *double* conversion.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.10 Malware Analyzer

A.3.6.2.10.1 Question 031 (AC Time, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that their *tool reported a list of program points influenced by untrusted input but did not find any influencable paths with significant time costs. Inspection of these points revealed that the time dependent operations are querying and CFG creation: Although, querying time is linearly dependent on number of samples.*
All querying operations are within the time budget for the provided payload size. The smallest add payload has to be at least 84 bytes, this limits the number of samples that could be added into DB, such that querying time is < 30 sec
Furthermore, while making CFGs, the program computes all entry points based on call targets, This can be used to create a multiplicative effect on the number of CFGs constructed for a given disassembly.
For ex:Following disassembly results in creation of 3 CFGs starting at: 0x0, 0x1 and 0x2 respectively.3: eb cs4: eb cs5: call 0x06: call 0x17: call 0x2
Each of the CFGs will have all the above instructions. However, input budget of 3000 bytes limits the maximum number of call instructions to 3000/11 ~= 300.Construction of 300 CFGs finishes well within the time budget of 30 sec.Thus the time of add_dasm command (for CFG creation) is also within the time budget.

Evaluation: The application did not contain an intended AC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.10.2 Question 035 (AC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that their *tool reported a program point where untrusted input influences the size of a list. Make CFG function i.e DasmHelper:make_function_cfg can be used to create a CFG where it ends up creating cascading basic-blocks. This can be exploited to get a summation effect using indirect jumps. I.e Given n instructions, the CFG function will*

create n basic blocks where each basic block is of size: n, n-1, n-2...1.
*Thus, when queried for function cfg (i.e get_cfg) the server will end up enumerating all instructions resulting in: n + n-1 + n-2 … = (n * (n+1)) /2 instructions. Although, only 16KB of this data is returned to user, the entire data will be stored in response.log whose size exceeds provided resource limit of 80KB.*

Evaluation: The challenge program contained an intended AC Space vulnerability in the packer detector functionality. Given a 256-integer feature vector, while calculating the average value heuristic an attacker can bypass the input guard to trigger an arithmetic exception. This exception is written to the log file. Normal log writes are overwritten, but the exception causes the entire stack trace to be appended to the log file. Northeastern identified the same potential vulnerability as Invincea. The exploit provided by Invincea exceed the resource usage limit of 80 KB.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.11 PowerBroker 1

A.3.6.2.11.1 Question 036 (AC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *since the attacker controls his own ID, and there is no check against the size of the ID, and his ID will be written to the log file on the remote side multiple times, the attacker can perform this attack by having a really long ID in his own .id file. In our PoC, we managed to generate log files of more than 2 MB on every participant other than the attacker himself by having a PDU of around 400 KB. In this challenge, our static analysis tool fails to identify any file write (since really there is no FileWriter at all) from the control flow graph, which led us to incorrect believe there is no such vulnerability. Later a manual reversing and inspection showed that a log file is generated, and we managed to perform the attack.*

Evaluation: The challenge program contains an AC Space vulnerability associated with the logging functionality. An attacker can connect to the victim, send the victim a connect message, then disconnect from the victim. The victim instance of the application will continue trying to reconnect with the attacker indefinitely adding to the log file. Northeastern identified another vulnerability in the log the log functionality of the challenge program. The application does not restrict the size of a user's ID, by creating an ID of "la"*390,000 a user id of 780,000 bytes is created. The stac.log file of the victim user contains the IDs of other users involved in a PowerBroker session and the malicious ID is written to each participant's log file. The observed resource usage of 1,565,244 bytes exceeds the resource usage limit of 1,000,000. The exploit has a user ID of 780,000 bytes leaving 20,000 bytes for establishing a connection. The vulnerability discovered by Northeastern is an in-scope AC Space vulnerability; however, since the challenge program contains an intended vulnerability, the impact is minimized. This vulnerability can be mitigated by enforcing a maximum size of a user ID.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.11.2 Question 050 (SC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *BigNumber.compareTo() has a small timing difference between different paths, and there is no blinding. However, I'm not sure if it is exploitable within the budget.*

Evaluation: The challenge program contained an intended SC Time vulnerability in the creation of a bid comparison message. This time is proportional to a user's bid amount. The method

identified by does not appear to exist within the challenge program. Additionally, there does not appear to be enough information in this response to indicate that Northeastern identified the intended vulnerability or an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.12 PowerBroker 2

A.3.6.2.12.1 Question 019 (AC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *since the attacker controls his own ID, and there is no check against the size of the ID, and his ID will be written to the log file on the remote side multiple times, the attacker can perform this attack by having a really long ID in his own .id file. In our PoC, we managed to generate log files of more than 2 MB on every participant other than the attacker himself by having a PDU of around 400 KB. In this challenge, our static analysis tool fails to identify any file write (since really there is no FileWriter at all) from the control flow graph, which led us to incorrect believe there is no such vulnerability. Later a manual reversing and inspection showed that a log file is generated, and we managed to perform the attack.*

Evaluation: The challenge program did not contain an intended AC Space vulnerability. Northeastern identified a vulnerability in the log the log functionality of the challenge program. The application does not restrict the size of a user's ID, by creating an ID of "la"*390,000 a user id of 780,000 bytes is created. The stac.log file of the victim user contains the IDs of other users involved in a PowerBroker session and the malicious ID is written to each participant's log file. The observed resource usage of 1,565,244 bytes exceeds the resource usage limit of 1,000,000. The exploit has a user ID of 780,000 bytes leaving 20,000 bytes for establishing a connection. The vulnerability discovered by Northeastern is an in-scope AC Space vulnerability. Because the challenge program does not contain an intended vulnerability, the answer key will be modified to reflect this change. This vulnerability can be mitigated by enforcing a maximum size of a user ID.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.12.2 Question 056 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: Northeastern responded that *BigNumber.compareTo() has a small timing difference between different paths, and there is no blinding. However, I'm not sure if it is exploitable within the budget*.

Evaluation: The challenge program did not contain an intended SC Time vulnerability. There does not appear to be enough information in this response to indicate that Northeastern identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.13 PowerBroker 3

A.3.6.2.13.1 Question 034 (SC Time, Intended Vulnerable, Answered No)

Response: Northeastern responded *N*

Evaluation: The operational definition requires sufficient justification for both yes and no responses. Northeastern provided no justification for this response.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.14 RSA Commander

A.3.6.2.14.1 Question 033 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that their *tool reported that untrusted input influences a loop in client connection handling. Manual inspection revealed that the server code does not properly handle closed connections, therefore a malicious client can cause the server's code to enter in an infinite loop. The code locations where improper validation happens are in the Communication class, when the "handle" function of the objects handshakeHandler or requestHandler is called. Moreover, the function HANDLER_STATE handle(InputStream is, PacketBuffer packetBuffer) returns HANDLER_STATE.WAITING when the connection is interrupted (because the function isPackedStillOk always returns true).*
Therefore closing the "sending" side of a connection makes the server code enter an infinite loop. After this operation, the client will never receive any communication back from the server and a server's thread will enter an infinite loop.
This behavior can be triggered, for instance, with Python code that first partially sends the first message of the handshake and then executes s.shutdown(socket.SHUT_WR) (where s is a socket instance connected to the server)

Evaluation: The challenge program contains and AC Time vulnerability in decrypting sent messages. An attacker can cause a benign user's instance of the application to trigger the slow mode of decryption by setting the counter value to 0. Northeastern and Draper identified an unintended vulnerability in the handshake procedure of the challenge program. When userA is attempting to send a message to userB, userA initiates the handshake procedure with userB. During the handshake procedure, the Communications.doHandshake() method calls the HandshakeHander.handle() method with a while loop that only terminates when HandshakeHander.handle() != HANDLE_STATE.WAITING. HandshakeHandler.handle() returns HANDLER_STATE.WAITING if isPacketStillOK() returns true. As implemented in HandshakeHandler.isPacketStillOK(), this method always returns true. If userB provides a partial response to userA's handshake request, the Communications.doHandshake() method will be stuck waiting for the completion of the message. While the victim user must initiate communication with the attacker, this represents an AC Time vulnerability by the strictest interpretation of the current operational definition.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.14.2 Question 040 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: Northeastern responded that their *tool reported that secret data is used to decide the execution of paths with potentially inequal running times in its cryptographic routines. We found that this software uses the Java BigInteger implementation of modular exponentiation. We verified that this implementation is not constant-time, specifically, on the reference platform, given two exponents e1 and e2, with e2 containing one more bit set to 1 than e1, the execution of (b^e2 mod p) takes about 0.05ms more than (b^e1 mod p).*
An attacker can use this information to analyze the time needed by the application to encrypt a set of chosen plain text messages and recover the private exponent of the public key used during a previous conversation between the victim's client and the server. Knowing the private exponent, an attacker can recover the symmetric key exchanged during the victim's previous communication and therefore decrypt the conversation.
Please also note that the application uses weak encryption algorithms. Specifically it uses by

default a 512bit rsa key (which is factorable in a reasonable amount of time) and it does not allow users to use larger keys. Also, it uses DES for symmetric encryption, which has been proved to be vulnerable. However, we consider these security issues as outside the scope of this engagement.)

Evaluation: The challenge program does not contain an intended SC Time vulnerability. Northeastern reported a vulnerability in the *java.math.BigInteger* implementation of modular exponentiation. The challenge program uses both the *BigInteger.modPow*() and the *BigInteger.pow()* and *BigInteger.mod()* implementations of modular exponentiation conditioned on the user input. Northeastern reported a timing difference of *0.05 ms* depending on whether a given bit of two otherwise identical exponents is set (1) or not set (0). It is unclear how this information can be used to craft a message that allows the attacker to segment and test each bit of the exponent individually. Given the noise on the reference platform in the current network configuration, the noise from the scheduler may make the overlap of the distributions of set and not set bits large. Given the operational budget of 10,000 operations, that it is unclear how an attacker would segment the exponent value, and the noisiness of the measurements on the reference platform, we do not believe that the challenge program is vulnerable.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.15 SmartMail

A.3.6.2.15.1 Question 015 (AC Time, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that *no vulnerable loops or recursion found. No complex data structures being used that could be abused for ACT attacks. MIME parsing including multipart works without an issue. By specifying a lot of addresses via To and CC (via Header injection on subject) and a heavily nested multipart message, runtimes of 6-7 seconds are possible (of which 3-5s come from To/CC).*

Evaluation: The challenge program did not contain an intended AC Time vulnerability

Post-Engagement Analysis Ruling: Correct

A.3.6.2.15.2 Question 024 (AC Space, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that *only one significant change in memory observed by deeply nesting MIME multi-part messages which will be serialized and deserialized. Those requests increase memory usage by 4MB, however they require a PDU of about 1KB. Therefore, only 10 requests can be made within the budget, thus only increasing memory usage by 40MB instead of 1024MB.*

Evaluation: The challenge program did not contain an intended AC Space vulnerability. Northeastern's response indicates a potential vulnerability given a sufficient budget.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.15.3 Question 039 (SC Time/Space, Intended Vulnerable, Answered No)

Response: Northeastern responded that t*he tool flagged suspicious use of cryptography in the application. We found that AES is used with custom padding, which leaks some information on the length of the administrator's email address (the accuracy of the length's estimate increases with the number of emails sent to the mailing list). However, due to the random length of the padding and the limited tries of 200, it leaks insufficient information as the exact length and*

significantly more tries would be necessary to go from the length to an email address. Both the length of To addresses is limited before processing the email and, although the MIME header can be overwritten through the Subject line and To/CC/BCC can be added, more than 200 tries are necessary for strictly larger than 99% accuracy.

Evaluation: The challenge program contained an SC Time/Space vulnerability which leaked the administrator of a given mailing list. The vulnerability was in the log file writes that occur when a message is sent. The set of all recipients of a given message (which includes the mailing list administrator and each of the subscribers) is alphabetized and written to a log file with the mailing list subscribers written to one log file and the administrator written to another. Using this side channel an attacker can run a binary search for each character and determine each subsequent character of the administrator's email beginning at the first character. Northeastern did not identify this vulnerability within the challenge program.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.16 Tweeter

A.3.6.2.16.1 Question 017 (AC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *fuzzing revealed that creating a user takes around 1K of PDU. Creating a user with avatar size=512 takes up >1MB (512x512 BMP). Therefore, by creating >1500 users we can reach >1500MB memory used.*

Evaluation: Northeastern identified the intended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.16.2 Question 049 (AC Time, Intended Vulnerable, Answered No)

Response: *Northeastern responded that no request to Tweeter takes up significant timing resources meeting the minimum bar. The closest to being ACT is forcing as many hashtag lookups as possible, however, it does not meet the bar. Similarily, forcing hashCode collisions on the hashtag set does not meet the minimum delayed response time (up to 3 minutes on NUC, which is less than the required 2500 seconds).*

Evaluation: The challenge program contained an AC Time vulnerability in the spell-correcting feature. The spell check algorithm's runtime is a function of word length and number of mistakes; however, the application has a maximum for word length and for number of mistakes. By providing a word which is within the applications maximum word limit (26 characters) and contains just enough mistakes to trigger the spell check algorithms slow execution time, the resource usage limit can be exceeded.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.17 WithMi 1

A.3.6.2.17.1 Question 007 (AC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *the problem is exposed during fuzzing stage: repeatedly sending files via command "sendfilezlib N" will take an excessive amount of memory which is never freed afterwards. However, due to the time constraint, I didn't look into the program and analyze the root cause of such a behavior. The PoC sends 178815 bytes to make the memory usage goes beyond 600 MB.*

Evaluation: The challenge program did not contain an AC Space vulnerability. It however does implement the Huffman de-compression algorithm which causes an intended AC Time vulnerability. It is possible that within the input budget the Huffman compression vulnerability could be used to trigger and AC Space vulnerability as well. The AC Team confirmed this in-scope AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.17.2 Question 010 (SC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *we took the challenge and executed it together with tcpdump to monitor the traffic. When talking in a private chat, packets are only exchanged among all users partificated in the chat. Therefore an attacker can easily figure out how many users are involved by observing how many IP addresses/ports there are in all from observing the traffic, which yields the number of participants of the private chat.*

Evaluation: The challenge program contained an intended SC Space vulnerability: when a new user is added to a WithMi session, the user receives a chat message larger than all other WithMi messages. The size of this message is proportional to the number of users in the chat. Northeastern identified a potential unintended vulnerability which is to identify the number of unique IP address and port pairs and map this to the number of users in a chat. The challenge question does not restrict the victim user to a single chat. Thus, the pairing of host and port values is not sufficient to determine the number of users in each chat.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.17.3 Question 053 (SC Space, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that *by fuzzing file sending commands, no such side channel is observed. Analyzing the program shows function FileTransfer.cramFile() is adding random padding bytes to the file data, which adds enough randomness to the file traffic that essentially prevents side channel leaks on file sizes.*

Evaluation: The challenge program did not contain an intended SC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.18 WithMi 2

A.3.6.2.18.1 Question 016 (AC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *the problem is exposed during fuzzing stage: repeatedly sending files via command "sendfilezlib N" will take an excessive amount of memory which is never freed afterwards. However, due to the time constraint, I didn't look into the program and analyze the root cause of such a behavior. The PoC sends 178815 bytes to make the memory usage goes beyond 600 MB.*

Evaluation: The challenge program contained an intended AC Space vulnerability in its implementation of Huffman de-compression. This vulnerability is triggered by consecutive 0's in a file. This triggers an exponential expansion in the de-compressed representation and results in exceeding the resource usage limit. Northeastern identified an unintended vulnerability which the AC Team confirmed.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.18.2 Question 029 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *problem is exposed during fuzzing stage: repeatedly sending files via command "sendfilezlib N" will take an excessive amount of memory which is never freed afterwards. However, due to the time constraint, I didn't look into the program and analyze the root cause of such a behavior. The PoC sends 178815 bytes to make the memory usage goes beyond 600 MB.* Northeastern reported that this POC could be used to exceed the time resource usage limit.

Evaluation: The challenge program contained an intended AC Time vulnerability in its implementation of Huffman de-compression. There are two “bugs” in the implementation that result in this vulnerability the first is in the readBoolean method in the BinaryIn class. This method allows reading beyond the end of the stream this coupled with two broken input guards can be used to exceed the limit on the number of characters to be de-coded. Northeastern discovered an unintended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.19 WithMi 3

A.3.6.2.19.1 Question 047 (SC Time, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that *as pointed out in our path cost analysis, there should be a small timing difference between two paths inside the inner loop of MontgomeryMultiplier.exponentiate(). However, further manual inspection shows that the EncryptionPrivateKey.decrypt() function implements blinding securely, which makes it impossible to exploit the timing side-channel*.

Evaluation: The challenge program does not contain an intended SC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.19.2 Question 055 (AC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *in CoolCompression.unzip(), num (which is returned by function readInt(stream)) can be a negative number, which essentially escapes the maximum size limit by the condition check of (totalSize > MAX_SIZE). An attacker can thus construct a special request that writes an excessive amount of data to the unzipped file. The loop was among the ones that are pointed out by our static loop analysis, and then we manually inspected it to confirm the finding*.

Evaluation: Northeastern identified the intended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.20 WithMi 4

A.3.6.2.20.1 Question 027 (SC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *in ParticipantConductor.addParticipantToParticipantHistory() method, the variable previouslyConnected (together with another variable) dominates three paths, and static path cost analysis shows there should be a noticeable difference in execution time between two of them, which is because method storeParticipant() eventually goes into file access, while the other path with previouslyConnected == true only performs a write to System.out. Further manual investigation shows observations can be made at the time when the add participant message is*

received, and at the time a receipt is sent back (by calling sendReceipt() in addParticipantToParticipantHistory() method), and luckily, the difference in execution time can be reflected in the observation of time elapsed between the two events. Due to time constraint, we assumed that the difference of path costs is very observable and didn't bother to PoC this question.

Evaluation: Northeastern identified the intended vulnerability in the challenge program

Post-Engagement Analysis Ruling: Correct

A.3.6.2.20.2 Question 043 (SC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *this one is almost the same as Question 047, but without blinding implemented. Theoretically it is possible to exploit the timing side-channel in mgMultiply() (the extra reduction).*

Evaluation: UMD identified an unintended vulnerability in the RSA implementation in this challenge program. Northeastern's response references the vulnerable Montgomery multiplication implementation.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.20.3 Question 045 (AC Space, Intended Vulnerable, Answered No)

Response: Northeastern responded that *in AwesomeSmashing.unzip(), num has to be greater than 0 in order to conduct a write to output stream. The loop in AwesomeSmashing.unzip() shows up in static loop analysis, since our analysis could not resolve the number range of num. Manual inspection made us believe it is not vulnerable to space complexity attack in that function. Also we manually analyzed other paths and concluded that no other path can lead to a file write.*

Evaluation: The challenge program contained an intended AC Space vulnerability. The challenge program implements run length encoding as part of its decompression scheme. This compresses consecutive identical bits within a file. The implementation in this challenge program contains a broken input guard which allows a user to bypass the maximum size enforced by the program. The number of bytes de-compressed is tracked using an integer therefore an attacker can cause this variable to overflow bypassing the input guard and triggering an AC Space vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.20.4 Question 051 (SC Space, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that *by analyzing FileTransfer.send() function, we learned that file content is directly coming from FileInputStream(), and directly put into the compression implementation. Since compressing the five files provided with the challenge yields different sizes, we believe there is likely to be a side channel in space: by observing how many bytes are sent to the other user. We further verified it by sending different files in the controlled environment via file-sending commands, and found it was indeed the case.*

Evaluation: Northeastern identified the intended vulnerability within the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.21 WithMi 5

A.3.6.2.21.1 Question 001 (AC Space, Intended Not Vulnerable, Answered Yes)

Response: Northeastern responded that *the same PoC in Question 007 proved that it is possible to use more than 600 MB of memory with less than 180KB of input. Again, due to the time constraint, I didn't look into the reason.*

Evaluation: The challenge program did not contain an intended AC Space vulnerability. Northeastern identified an in-scope unintended AC Time vulnerability confirmed by the AC Team.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.21.2 Question 026 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: Northeastern responded that *we took the challenge and executed it together with tcpdump to monitor the traffic. When talking in a private chat, packets are only exchanged among all users partificated in the chat. Therefore an attacker can easily figure out how many users are involved by observing how many IP addresses/ports there are in all from observing the traffic, which yields the number of participants of the private chat.*

Evaluation: The challenge program did not contain an intended SC Space vulnerability. Northeastern discovered an unintended vulnerability which is to identify the number of unique IP address and port pairs. This number maps to the number of users in a chat. The challenge question does not restrict the victim user to a single chat. Thus, the pairing of host and port values is not sufficient to determine the number of users in each chat.

Post-Engagement Analysis Ruling: Incorrect

A.3.6.2.21.3 Question 032 (SC Time, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that *in ParticipantConductor.addParticipantToParticipantHistory() method, the variable previouslyConnected (together with another variable) dominates three paths, and static path cost analysis shows there should be a noticeable difference in execution time between two of them, which is because method addParticipantToParticipantHistoryUtility() eventually goes into file access, while the other path with previouslyConnected == true only performs a write to System.out. However, such a difference is not reflected in observation, since the receipt is sent prior to addParticipantToParticipantHistoryUtility() is called. The two different paths controlled by variable previouslyConnected do not have a significant execution time difference. Hence we believe the side channel is not exploitable (or rather, there is no timing side channel to an external observer.*

Evaluation: The challenge program did not contain an intended SC Time vulnerability

Post-Engagement Analysis Ruling: Correct

A.3.6.2.22 WithMi 6

A.3.6.2.22.1 Question 054 (AC Time, Intended Vulnerable, Answered Yes)

Response: Northeastern responded that the s*ame PoC in Question 007 proved that it is possible to trigger the complexity in time. But again, I'm not sure it is due to an algorithmic complexity.*

Evaluation: The challenge program did not contain an intended AC Time vulnerability. Northeastern identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.6.2.22.2 Question 057 (AC Space, Intended Not Vulnerable, Answered No)

Response: Northeastern responded that *the PoC in Question 007 didn't show a time complexity in this one.*

Evaluation: The challenge program did not contain an intended AC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.7 University of Maryland

A.3.7.1 UMD Overview

UMD answered 12 questions with a 58% accuracy. UMD identified 5 unintended vulnerabilities in CyberPoint challenge programs, 4 referring to the RSA implementation.

Table A-55: Engagement 3 UMD Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	0	0	0	0
SC Time	8	6	75	5	63
SC Space/ Time	0	0	NA	0	NA
AC in Space	2	2	100	1	50
AC in Time	1	1	100	1	100
Total	**12**	**9**	**50**	**7**	**58**

Table A-56: Engagement 3 UMD Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	8	8	**100**	6	**75**
Not Vulnerable	4	1	**25**	1	**25**
Yes Answer	11	8	**73**	6	**55**
No Answer	1	1	**100**	1	**100**

A.3.7.2 UMD Specific Responses

A.3.7.2.1 Airplan 3

A.3.7.2.1.1 Question 025 (AC Time, Intended Vulnerable, Answered Yes)

Response: UMD responded that: w*e decompiled the program. We performed static taint analysis that produced the program depdence graph. We can query the secrets and taints on this graph. We also ran static bound analysis overnight for this program. We found that the server responds to the capacity quries. The class net.cybertip.scheme.Capacity contains an*

implementation of the Ford-Fulkerson algorithm whose worst case scenario could be O(Ef), where f is the capacity of the network. The server also doesn't cache any previous computation results, so when the worst-case query is repeated, the server can be very slow to respond them.

Evaluation: UMD identified the intended AC Time vulnerability within the challenge program

Post-Engagement Analysis Ruling: Correct

A.3.7.2.2 Airplan 4

A.3.7.2.2.1 Question 030 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: UMD responded that: *we decompiled the program and ran static taint analysis on it. We found there is a function reachable that will test if two airports are connected. If we can observe a request from an authenticated client to see if (A,B) are connected, the response size will be different if (A,B) are connected vs if they are not.*

Evaluation: This challenge program does not contain an intended SC Space vulnerability. All graph properties packets are padded to exactly 19 characters as to not leak any information

Post-Engagement Analysis Ruling: Incorrect

A.3.7.2.3 BidPal 1

A.3.7.2.3.1 Question 014 (SC Time, Intended Vulnerable, Answered Yes)

Response: UMD responded that: *we decompiled the program. We executed the example scripts with the jProfiler profiling tool, which provided us hotspot methods and call tree that helped us locate the methods influenced by secret and taint values. We also ran the bound analysis on all the methods in this program. The Montgomery multiplication routine (i.e., edu.computerapex.math.MgMultiplier.mgMultiply) leaks information about the secret key since the exponentation algorithm uses mongomry form. When using this form, a montgomery extra reduction (which takes more time) happens based on the distribution of values inside the algorith. By observing when the reduction happens, we can learn informaion about the distirbution of internal values and thus leak the key. The paper titled "Improving Divide and Conquer Attacks Against Crypto systems by Better Error Detection / Correction Strategies" discusses the literature.*

Evaluation: The challenge program uses Montgomery multiplication in its exponentiate method with the exponent coming from the user's private key. As noted in the reference paper, this results in an extra reduction which can leak the private key. The paper referenced by UMD contains an exploit which can be used against this RSA implementation.

Post-Engagement Analysis Ruling: Correct

A.3.7.2.4 BidPal 2

A.3.7.2.4.1 Question 037 (SC Time, Intended Vulnerable, Answered Yes)

Response: UMD responded that: *we decompiled the program and executed the scripts with the jProfiler turned on. We found early exit in org.techpoint.sale.messagedata.BarterMessageData.verifySpacing routine that tests bids against highest / other bids along with involving users private key.*

Evaluation: The challenge program contains an SC Time vulnerability in the org.techpoint.sale.messagedata.OfferAnalysisData.compose method. This method iterates from 0 to the maximum possible bid performing an expensive operation at each iteration. When the user's bid is reached, the formGuide() method is called to begin generating the bidComparison message. The time between the creation of the bidReciept and the bidComparison messages leaks the user's bid. UMD identified the verifySpacing method as vulnerable. The class identified by UMD BarterMessageData does not contain a verifySpacing method, the OfferAnalysisData class in the same package does. The verifySpacing method does not leak any information about the user's bid.

Post-Engagement Analysis Ruling: Incorrect

A.3.7.2.5 Collab

A.3.7.2.5.1 Question 042 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: UMD responded that: *we decompiled the program. We profiled the program and ran dynamic taint analysis. We found that in the BTree, add method takes a differing amount of time to split a tree node depending on how full it is, if it has a secret/auditor node as a subnode then the time will occur when you don't expect it to, which can be exploited to reveal the value of the secret.*

Evaluation: This challenge program did not contain an intended SC Time vulnerability. The *split(DataNode orignode)* method reference by UMD performs 3 log writes and part of its implementation resulting in an SC Time vulnerability. This side channel is non-locally balanced in the SchedulingSandbox.add() method resulting in a balanced insert operation

Post-Engagement Analysis Ruling: Incorrect

A.3.7.2.6 PowerBroker 3

A.3.7.2.6.1 Question 034 (SC Time, Intended Vulnerable, Answered Yes)

Response: UMD responded that: *we decompiled the program. We executed the example scripts with the jProfiler profiling tool, which provided us hotspot methods and call tree that helped us locate the methods influenced by secret and taint values. Montgomery multiplication routine (net.roboticapex.algorithm.FastModularMultiplier.fastModularMultiply) leaks information about the secret key by a montgomery extra reduction. See the paper "Improving Divide and Conquer Attacks Against Cryptosystems by Better Error Detection / Correction Strategies".*

Evaluation: The paper referenced by UMD contains an exploit which can be used against this RSA implementation.

Post-Engagement Analysis Ruling: Correct

A.3.7.2.7 PowerBroker 4

A.3.7.2.7.1 Question 011 (SC Time, Intended Vulnerable, Answered Yes)

Response: UMD responded that: *we decompiled the program and profiled the executions. Montgomery multiplication routine (i.e., com.virtualpoint.numerical.MgMultiplier.mgMultiply) leaks information about the secret key by a montgomery extra reduction. Please refer to the paper ""Improving Divide and Conquer Attacks Against Cryptosystems by Better Error Detection / Correction Strategies"" for the details.*

Evaluation: The challenge program contains an intended SC Time vulnerability. The RSA implementation uses a combination of Montgomery multiplication and Chinese Remainder theorem to perform modular exponentiation efficiently. This results SC Time vulnerability where the plot of attacker-controlled base vs response time during connection establishment forms a sawtooth wave where the frequency of the wave is the equal to the smaller of the two RSA primes for a given user. The keys to this side channel are two conditionals: one within the Montgomery multiplication resulting in an extra reduction and the other in the exponentiation method conditioned on the individual bits of the exponent (private key). UMD identified the extra reduction conditional and the paper referenced contains an exploit which can be used on the RSA implementation in this challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.7.2.8 RSA Commander

A.3.7.2.8.1 Question 040 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: UMD responded that: *we decompiled the program. We also ran static taint anlaysis that produces the program dependence graph. We found that the program Decrypt-then-MAC, checking message integrity after decrypting in RequestPacketParser.parse allows for recovering the key.*

Evaluation: The challenge program does not contain an intended SC Time vulnerability. The potential vulnerability identified by UMD does not appear to allow for the full extraction of the secret.

Post-Engagement Analysis Ruling: Incorrect

A.3.7.2.9 WithMi 3

A.3.7.2.9.1 Question 047 (SC Time, Intended Not Vulnerable, Answered No)

Response: UMD responded that: *we decompiled the program and executed it with profiling. The possible leakage in net.robotictip.numerical.OptimizedMultiplier is masked beyond the bounds of the operational budget. Note that there might be an issue with the DH used. This is since the montgomery ladder + weak multiplication routine leaks the key. This is code from Eng2 which is considered secure but I (Daniel) think it isn't.*

Evaluation: The challenge program does not contain an intended SC Time vulnerability

Post-Engagement Analysis Ruling: Correct

A.3.7.2.9.2 Question 055 (AC Space, Intended Vulnerable, Answered Yes)

Response: UMD responded that: *“cool” compression decompress bounds output to 100MB, but not zlib, see "zip bomb." Is this a fault of the library? I think not, because the application code could check the size the same way “cool” does.*

Evaluation: The challenge program contains an AC Space vulnerability in its decompression implementation. The decompression algorithm uses run length encoding to compress and de-compress files sent. Compression is performed by identifying consecutive identical bits. The number of consecutive identical bits to be decompressed is an attacker controlled value. The check on this value starts at the attacker provided amount and counts down stopping when the value is equal to 0. An attacker can set this value to a negative number to exceed the resource usage limit. The AC Team confirmed the unintended vulnerability identified by UMD.

Post-Engagement Analysis Ruling: Correct

A.3.7.2.10 WithMi 4

A.3.7.2.10.1 Question 043 (SC Time, Intended Vulnerable, Answered Yes)

Response: UMD responded that: *we decompiled the program and executed with profiling. We also ran bound anlaysis of the program. Montgomery multiplication routine (i.e., in com.digitalpoint.math.MgProductGenerator) leaks information about the secret key by a montgomery extra reduction. Please see the paper "Improving Divide and Conquer Attacks Against Cryptosystems by Better Error Detection / Correction Strategies".*

Evaluation: This challenge program does not contain an intended SC Time vulnerability the paper referenced by UMD contains an exploit which can be used against this RSA implementation.

Post-Engagement Analysis Ruling: Correct

A.3.7.2.10.2 Question 045 (AC Space, Intended Vulnerable, Answered Yes)

Response: UMD responded that: *"awesome" compression bounds to 100MB, but not zlib.*

Evaluation: The challenge program contained an intended AC Space vulnerability. The challenge program implements run length encoding as part of its decompression scheme. This compresses consecutive identical bits within a file. The implementation in this challenge program contains a broken input guard which allows a user to bypass the maximum size enforced by the program. The number of bytes de-compressed is tracked using an integer therefore an attacker can cause this variable to overflow bypassing the input guard and triggering an AC Space vulnerability. There is not enough information in UMD's response to indicate that they identified this vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.3.8 University of Utah

A.3.8.1 Utah Overview

Utah answered 4 questions with a 75% accuracy focusing solely on algorithmic complexity questions.

Table A-57: Engagement 3 Utah Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	0	0	NA	0	NA
SC Time	0	0	NA	0	NA
SC Space/ Time	0	0	NA	0	NA
AC in Space	2	2	100	1	50
AC in Time	2	2	100	2	100
Total	**4**	**4**	**100**	**3**	**75**

Table A-58: Engagement 3 Utah Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	4	4	**100**	3	**75**
Not Vulnerable	0	0	**NA**	0	**NA**
Yes Answer	4	4	**100**	3	**75**
No Answer	0	0	**NA**	0	**NA**

A.3.8.2 Utah Specific Responses

A.3.8.2.1 Airplan 2

A.3.8.2.1.1 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Response: Utah responded that their *tools detect suspect recursive functions like `Orderer.changingRank()` and flag them as potential hot spots. Unfortunately, this application was one of 6 (out of a total of 24 total apps) that triggered a bug in Soot that prevented this particular analysis from completing. We are still investigating this bug and whether we can fix or work around it.*

Fortunately, we had other tools that were not affected by this bug. One of those flags code involving sorting algorithms, which are frequent sources of vulnerabilities. Based on the results from that tool, we manually inspected `changingRank()`. This allowed us to determine that its worst-case runtime cost is bounded by a recurrence equation of the following form where `n` is the length of the list being sorted, `O(n)` is a linear function of `n` and `T(n)` is the total runtime of `changingRank` on a list of length `n`:

T(n) = T(n/6) + 2T(5n/6) + O(n) [when n mod 6 == 0]
T(n) = T(n/6) + T(5n/6) + O(n) [when n mod 6 != 0]

Though we are still in the process of developing tools to automatically extract and solve this sort of equation, a manual solution shows that it is super-linear in `n` and grows rapidly when the recurrence frequently goes through the first case rather than the second case. For example, `T(258)` will be very large.

From there we merely tracked backwards through the code to find inputs that are being sorted and whose size we can control. The attached file, `airplan2attack.txt`, is an input that satisfies this.

Evaluation: Utah identified the intended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.8.2.2 Info Trader

A.3.8.2.2.1 Question 002 (AC Time, Intended Vulnerable, Answered Yes)

Response: Utah responded that *by using the loop/recursion detecting tool we developed, which can show where the recursion happens and also the stack information when the recursion happens, we found several suspicious methods in the program that might trigger AC Time or AC Space vulnerability. We note that this is the same tool that failed to run on airplane_2 and linear_algebra_platform due to a bug in the Soot framework--although the tool failed to run on 6 apps, it ran to completion on 18 apps, including this one.*

For example, these are some results related to these vulnerabilities from our tool:

Found recursive calls on infotrader.dataprocessing.SiteMapGenerator.visitAllNodes, skip.
infotrader.messaging.controller.module.RunInfoTrader.main
infotrader.userinteraction.SitemapServlet.mainx
infotrader.userinteraction.DocumentParser.<init>
infotrader.dataprocessing.SiteMapGenerator.genSiteMap
infotrader.dataprocessing.SiteMapGenerator.serializeTree(1)
infotrader.dataprocessing.SiteMapGenerator.serializeNode
infotrader.datamodel.HyperLink.serialize
infotrader.dataprocessing.SiteMapGenerator.visitAllNodes
infotrader.dataprocessing.SiteMapGenerator.visitAllNodes(1) recursion begins
infotrader.dataprocessing.SiteMapGenerator.visitAllNodes recursion ends

Found recursive calls on infotrader.dataprocessing.SiteMapGenerator.serializeNode, skip.
infotrader.messaging.controller.module.RunInfoTrader.main
infotrader.userinteraction.SitemapServlet.mainx
infotrader.userinteraction.DocumentParser.<init>
infotrader.dataprocessing.SiteMapGenerator.genSiteMap
infotrader.dataprocessing.SiteMapGenerator.serializeTree(1)
infotrader.dataprocessing.SiteMapGenerator.serializeNode(1) recursion begins
infotrader.dataprocessing.SiteMapGenerator.serializeNode recursion ends

By looking at source code of suspicious methods, we observed that when uploading a document to the server, the server will regenerate the whole site map file by traversing the documents through links between them.

We constructed an attack that uploads a series of documents sequentially (the runnable attack program is in the directory attack_program). These documents are connected by two hyperlinks: for example, 1 -> [2, 3], 2 -> [3, 4], 3 -> [4, 5], etc. (the number represents the title of documentation). The serialization of hyper links therefore results in an exponential traverse. And, since the output XML file from the serialization also contains an exponential repetition of these hyperlinks, the logical size of file Sitemap.xml will exceed the usage limit.

Evaluation: The challenge program does not contain an intended AC Time vulnerability. Northeastern identified an unintended vulnerability in the challenge program. Utah identified a potential vulnerability in the document traversal algorithm similar to the one identified by Iowa. Utah's exploit creates a complex traversal structure to exceed the resource usage limit. Utah's

exploit was tested using 20 document resulting in a resource usage of 921.72 seconds. There is no doubt that given enough document upload the resource usage limit will be exceeded. Uploading 20 documents however, requires 5129 bytes which exceeds the input budget. Northeastern's exploit creates a long document chain with a loop at the end. This eventually triggers a *stackOverflowException* but not before the resource usage limit is exceeded. Utah flagged the document traversal algorithm as vulnerable and while their exploit was not valid, Northeastern's was.

Post-Engagement Analysis Ruling: Correct

A.3.8.2.2.2 Question 004 (AC Space, Intended Vulnerable, Answered Yes)

Response: Utah provided the same response for this question as Question 004.

Evaluation: The unintended AC Time vulnerability identified by Utah could potentially be used to trigger an unintended AC Space vulnerability. The challenge program contained a broken input guard which while intended to prevent cycling allowed users to upload a document with the same name as the root node. If an attacker uploads a document linking to the document with the same name as the root, the resulting document traversals that generate the sitemap xml file will not cause a StackOverflowError but will result in a sitemap xml file which exceeds the resource usage limit. The sitemap xml following the 20 uploads is 6,617 kB, which is less than the resource usage limit of 100,000 kB.

Post-Engagement Analysis Ruling: Incorrect

A.3.8.2.3 Linear Algebra Platform

A.3.8.2.3.1 Question 005 (AC Space, Intended Vulnerable, Answered Yes)

Response: Utah responded that *the simple test case below causes the program to attempt to create a matrix of size 20000 x 20000. It then exceeds the maximum allowed memory.*

{"numberOfArguments":2, "operation":2, "args":[{"rows":1, "cols":1, "matrix":"1.000000000000000\n"},{"rows":20000, "cols":20000, "matrix":"\n"}]}

How we found the attack:

One of our tools flags memory allocations as possible attack surfaces. Unfortunately, this application was one of 6 (out of a total of 24 total apps) that triggered a bug in Soot that prevented this particular analysis from completing. We are still investigating this bug and whether we can fix or work around it.

Nevertheless, one of our analysts found the allocation in the function MatrixSerializer.readMatrixFromCSV and noticed that most calls to this function set the condition enforceSize to true, but the ShortestPath operation calls this function with enforceSize set to false.

Upon examining how the function uses this condition, it was noted that this allows the user to bypass the size check on the second matrix in a ShortestPath operation.

Evaluation: The challenge program does not contain an intended AC Space vulnerability. Utah's response is the same as the one provided by GrammaTech, essentially the challenge

program uses the MatrixSerializer.readMatrixFromCSV() method to parse input matrices. This method has two enforcement parameters enforceWidth and enforceSize which are used all but one of the possible input matrices. The shortest path operation of the matrix takes two input matrices, an adjacency matrix and a targetNodes vector representing the set of nodes to find the shortest path between. The vector of nodes is parsed with the enforceWidth and enforceSize parameters set to false. The readMatrixFromCSV method proceeds to allocate memory for the targetNodes vector using the user submitted arguments which results in a memory usage of 3,653.928 MB exceeding the resource usage limit of 2,500 MB. An illegal argument exception is then triggered once the method finds that the number of rows in the input matrix and the number of rows declared by the user do not match; however, this is after the memory has been allocated for the targetNodes vector. This is an in-scope unintended vulnerability in the challenge program

Post-Engagement Analysis Ruling: Correct

A.3.9 Vanderbilt University

A.3.9.1 Vanderbilt Overview

Vanderbilt answered 12 questions with a 58% accuracy. For the question 28 in the Collab challenge program, Vanderbilt identified a vulnerability using out-of-scope observables. In the Tour Planner and SmartMail challenges (questions 38 and 39 respectively) Vanderbilt is very close to identifying the intended vulnerabilities; however, they missed key components of the challenge program which may have caused them to miss the intended vulnerabilities.

Table A-59: Engagement 3 Vanderbilt Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	5	3	60	3	60
SC Time	3	2	67	2	67
SC Space/ Time	1	0	0	0	0
AC in Space	1	1	100	1	100
AC in Time	2	1	50	1	50
Total	**12**	**7**	**58**	**7**	**58**

Table A-60: Engagement 3 Vanderbilt Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	6	3	**50**	3	**50**
Not Vulnerable	6	4	**67**	4	**67**
Yes Answer	5	3	**60**	3	**60**
No Answer	7	4	**57**	4	**57**

A.3.9.2 Vanderbilt Specific Responses

A.3.9.2.1 Airplan 1

A.3.9.2.1.1 Question 052 (AC Time, Intended Vulnerable, Answered No)

Response: Vanderbilt responded: *we used our Janalyzer static analysis tool to analyze the program for suspicious areas. From there, we identified the edu.cyberapex.order.Orderer.sort() method as an interesting method. We wrote a driver for Symbolic Pathfinder to analyze this method with symbolic strings, and we applied our worst-case analysis search algorithm. The generated data points showed that the sorting method was not vulnerable up to the size we were able to analyze. Because a lot of the calls go through this sorting routine, we belive the program may be non-vulnerable.*

Evaluation: The intended vulnerability in this application was in the shortest path determination. Given a route map, the application calculated the shortest path using A-star search with an inconsistent heuristic.

Post-Engagement Analysis Ruling: Incorrect

A.3.9.2.2 Collab

A.3.9.2.2.1 Question 018 (AC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt responded: *we do not believe that this example contains a time complexity vulnerability. The example contains two data structures for storing events: a Scheduling Sandbox and a B-tree. The scheduling sandbox is also a rebalancing tree (like the B-tree). We have analyzed both of these and their operations for adding new elements. These were the only methods that showed up as interesting in the Janalyzer. We analyzed the operations with SPF-WCA. For instance, for the B-tree add operation, we could scale the analysis to input size 120 (i.e. 120 symbolic elements added to the tree), and it showed the O(log n) complexity, thus it does not seem vulnerable. The same applies for the scheduling sandbox. See attached plots.*

Evaluation: The challenge program does not contain an intended AC Time vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.9.2.2.2 Question 028 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: Vanderbilt responded: *yes, we believe there is a side channel in space by which the user "picard" can discover the eventID relating to one of his audits, assuming he has at least one audit event on his calender. The side channel is located in the log files generated by application. Each time picard adds an event to his calender, the string "preparing to return add status" is usually added to the log at least 86 times (randomness in the number is due to the CollabServer.fileWrite() function but does not obscure the side channel). However, this is not always true! In the case where a status flag is set, nothing is added to the log file. By manual code inspection, we determined this flag to be set when trying to add an event to an already full node. This is when splitting the node becomes necessary.*

Each node of the BTree can store a fixed number of events (8). When a ninth event would be added to that node, the node is instead split, the flag is set and nothing is added to the log file. Importantly, the node stores both normal user events and audit events. picard can use this space side channel as follows.

The only events of his calender that picard cannot see are the auditor events. In fact, each event is associated with a single user and so all other events are actually those added by picard himself. picard can thus predict when adding an event would cause a node to split. The only time when his predition would be wrong is when there is something stored at a node that picard can not see -- an audit event!

Using this side channel allows picard to determine which node of the BTree the audit event is in. picard can then continue to narrow its eventID in a manner similar to binary search -- splitting the node into two even ranges at each step, determining which one contains the audit event and continuing until the eventID is ascertained.

Evaluation: This side channel in the challenge program requires an attacker to have access to observe the log-file writes on the server. The description and challenge question do not provide this as a valid observable.

Post-Engagement Analysis Ruling: Incorrect

A.3.9.2.2.3 Question 042 (SC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt responded: w*e do not believe that there is a side channel in time that allows picard to discover the eventID relating to one of his audits in 200 operations.*

By manual code inspection, we were able to find a small timing side channel in the calling of the collab.dstructs.objs.DataNode.takestep() method by the collab.SchedulingSandbox.getRange() method. In takestep(), an event is added to the search results only if it is a normal user event and not an auditor event but it is called for both kind of events. Therefore if there were 10 normal events and 1 audit event within a given search range, it would be called 11 times as opposed to the 10 times it would have been called when there is no audit event. However both cases would return the same set of results to the user.

While this is a side channel in time, the cost of the takestep() method is so small that it is not detectable. Thus we ruled out this potential side channel.

We also looked for side channels in time when adding an event. Since more work is done when a node is split during an add, we were curious as to whether the exploit found in Q028 could be reproduced using a timing side channel. However, the number of operations (200) allowed is far too low for this exploit to deterine the eventID of the audit event even in the case where the adds that split a node and those that don't are seperable with certainty. We also inspected the commit, quit and undo methods and found no exploitable timing side channel. While these results are not exhaustive, a thorough inspection of all possible user interactions yielded no exploitable timing side channel, leading us to conclude that the application has no side channel in time that can reveal the secret within the budget.

Evaluation: The challenge program does not contain an intended SC Time vulnerability. Vanderbilt's response shows that they identified the potential SC Time vulnerability in the add operation but recognized that is did not allow for the extraction of the secret value.

Post-Engagement Analysis Ruling: Correct

A.3.9.2.3 SmartMail

A.3.9.2.3.1 Question 024 (AC Space, Intended Not Vulnerable, Answered No)

Response: Vanderbilt responded: *we believe that this application does not contain a memory vulnerability. Our JAnalyzer tool points us to the method EmailEventsMapper.map(), because it is inside a 3 layer nested loop and allocates an object. The object is potentially large and escapes the method. The method is called from main->callMapper->callDumper, which iterates over the Collections in StateHolder.sfmap. That map is populated exclusively by the StateHolder.append() method. That method is called from SequenceFileWriter.writeEmail() and SequenceFileWriter.writeWord(). The latter of those is the most complex, containing two loops with a call to StateHolder.appen() inside. The latter one splits up the body text of an e-mail on space characters, so it seems the worst case memory consumption is triggered by a body that contains as many words as possible ("a a a a ..."). Additionally, the first loop iterates over receivers and also has a call to append() inside. We thus created an e-mail with as many receivers as possible and as many words as possible in both body and subject (see Vulnerable.sh). This e-mail does not increase memory consumption of the server to anywhere near the budget (1GB). We conclude that this program does not contain a memory vulnerability.*

Evaluation: The challenge program does not contain an intended AC Space vulnerability

Post-Engagement Analysis Ruling: Correct

A.3.9.2.3.2 Question 039 (SC Time/Space, Intended Vulnerable, Answered No)

Response: Vanderbilt responded: *there is a side channel in space which allows the attacker to detect, among other things, whether a mailing list contains a secret administrator or not. The attacker can obtain the list of (public) subscribers to a mailing list through an active operation. Then he can send an email to the list (another active operation) and observe the app's interaction with the log files (a passive operation). The 'addlog1.log' file grows if and only if there was a secret subscriber in the list, whereas public recipients are only logged to the 'addlog2.log' file. Thus, the attacker can look at the logs and infer whether or not a secret administrator is subscribed to that list.*

Some other interesting facts: A logfile grows each time recipients of the corresponding type are included in the message that is sent through SmartMail. But due to random padding, the encrypted logged content is not always the same -- in fact, due to urlencoding, its length in bytes can vary, even when repeatedly sending out the same email to the same recipient. However, if we urldecode the content added to the logfile, the size in bytes of the urldecoded content IS always the same. Thus, an attacker with access to the logfiles can count the number of recipients added to a logfile.

Although the above side channel in space is sufficient for an attacker to detect whether a list contains a secret member, it is not enough to extract the username of the secret user, which is the goal in this question. To achieve that, the attacker would need to exploit a side channel (possibly a time side channel, as the question mentions one in space AND time) -- for instance, carrying out a segmented-oracle-like attack on the secret username (e.g., via the send-mail or get-mail actions), or discovering the ""thekeyisbigenoug"" AES encryption key to decode the address stored in the logfile.

The low operational budget (only 200 operations) and high required probability of success (99%) make it extremely unlikely for an attacker to be able to exploit a side channel to achieve the above. EmailSenderReceiver limits email address left-hand sides to 25 lowercase characters, and the AES encryption key is 16 lowercase characters. However, a segmented oracle approach on a secret of length 16, with 26 different lowercase characters to choose from, may require 416 attempts even in complete absence of any noise. A budget of 200 active operations is not enough to extract the secret username (or secret AES key).

Evaluation: The challenge program contained an SC Time/Space vulnerability which leaked the administrator of a given mailing list. The vulnerability was in the log file writes that occur when a message is sent. The set of all recipients of a given message (which includes the mailing list administrator and each of the subscribers) is alphabetized and written to a log file with the mailing list subscribers written to one log file and the administrator written to another. Using this side channel an attacker can run a binary search for each character and determine each subsequent character of the administrator's email beginning at the first character. Vanderbilt's response indicates that they have identified part of the vulnerability particularly in the log write functionality however they did not identify the determinism in the order of log-writes. Additionally, Vanderbilt is attempting to implement a segmented oracle exploit. If they recognized the predictability in the timing of log writes, they may have identified the ability to binary search each individual character.

Post-Engagement Analysis Ruling: Incorrect

A.3.9.2.4 Tour Planner

A.3.9.2.4.1 Question 003 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: Vanderbilt responded: *using the space side channel and known information, we can narrow down the set of potential city name combinations to a manageable number (below 3000), and then brute-force-query the oracle (within budget) to find the right combination.*

Although their payload is encrypted, the length of the TCP packets that contain a tour query request constitute a side channel in space. Using active operations (no cost) and packet sniffing, we can examine the protocol and find a point in the TCP conversation at which the size of a particular packet sent from client to server grows in direct proportion to the number of bytes in the request URL, including the query string in it. (Using the provided curl-based query script, this happens at the 17th IP packet). Thus, while we cannot see the plaintext URL because it's encrypted, we can figure out its length. Since we know the shape of the URL, the number of cities, the length of the '&point=' strings, etcetera, we can infer the sum of the lengths of all five city names. So the secret is the five names, and the observable is the sum of their lengths.

We obtain the list of cities (an active operation). Then we measure their lengths and see that the frequency distribution is {3:1, 4:2, 6:6, 7:4, 8:5, 9:4, 10:2, 11:1}. For this multiset of lengths, the minimum possible sum is 15 (5 times 3), and the maximum possible sum is 55 (5 times 11). Consider a set of constraints over a, b, c, d, e (the five unknown lengths) and T, which denotes the known parameter (their total sum). We need one constraint asserting that a+b+c+d+e=T, and additional constraints asserting that each of the five variables is in the set of valid lengths (e.g., a is_in {3,4,6,7,8,9,10,11}, b is_in {3,4,6,7,8,9,10,11}, etcetera).

We generate 41 sets of constraints, one for each possible value of T in [15, 16,.., 55]. Finally, we use the ABC model counter to count the number of solutions to each set of constraints. The respective model counts are: [1, 5, 10, 15, 30, 56, 85, 130, 200, 285, 385, 515, 680, 855, 1040, 1251, 1460, 1650, 1825, 1980, 2091, 2140, 2145, 2100, 1995, 1846, 1665, 1460, 1235, 1010, 801, 610, 445, 310, 205, 126, 70, 35, 15, 5, 1]. Since the maximum is 2145, even in the worst case (when the total sum is 37) we will need to try no more than 2145 possible combinations of cities. Since our operational budget is 2999 after the passive operation, we can afford up to 2999 oracle queries. Therefore, we can always find the right combination of cities.

NOTE: The above calculation is inaccurate due to accidental mishandling of two city names that contain spaces. After fixing that, the distribution is {4:1, 6:6, 7:4, 8:5, 9:4, 10:3, 11:2}; the minimum sum is 5x4=20; the maximum sum is still 5x11=55; the model counts are {1, 0, 5, 5, 15, 25, 45, 75, 115, 180, 251, 355, 470, 605, 750, 896, 1040, 1155, 1250, 1300, 1306, 1265, 1180, 1060, 910, 751, 590, 440, 310, 205, 126, 70, 35, 15, 5, 1}; and the worst case is 1306 oracle queries.

Evaluation: The challenge program does not contain an intended SC Space vulnerability. The challenge question restricts the number of cities to 5. This means there are 25 chose 5 = 53,130 possible sets of cities a user can submit. Vanderbilt's set of possible sets of user queries only contains 16807 counts. Additionally due to the implementation of AES256 with a block size of 128 bits, there are 24696 collisions in the worst case and not enough oracle queries to resolve this.

Post-Engagement Analysis Ruling: Incorrect

A.3.9.2.4.2 Question 038 (SC Time, Intended Vulnerable, Answered No)

Response: Vanderbilt responded: a*ny user choice of five different cities will yield very similar timings. (For simplicity's sake, let us assume that the uses picks five different cities. This can be assumed without loss of generality, since a user choice that repeats the same city multiple times would be a special case, and an attack that relies on the occurrence of such a particular case would not satisfy our operational definition of a side channel.)*

Once the number of cities is set to exactly 5, their names or positions do not really affect the route calculation in a significant way -- certainly not in one that can be detected within the operational budget.

As a profiling experiment, we first measured the total time (between the first packet of a user's request and the last packet of its response) as monitored by a potential attacker on the reference platform (NUC). We did this for 50 different route requests, each of them a random sample of 5 cities. Each sample of 5 cities was requested 20 times and the mean time was calculated (for a total of 1000 requests). As the box plot shows, there is too much overlap -- the (nearly) same observable maps to dozens of different secrets. The overall mean is 173.25 milliseconds, and over 90% of all observations lie between 173.25 and 173.5 milliseconds (a 250 microsecond range). Total timing between first and last packet is not a viable way to separate five-city queries in order to obtain five arbitrary city names.

One could also look into finer-grained time-observables, such as deltas between events in the packet exchanges; for instance, the time deltas between progress reports (which are sent out

every time the spanning tree algorithm adds a new vertex to the tree), or between the last progress report and the beginning of the JSON response. However, any difference between these observables is bounded by the aforementioned (less than a quarter of a millisecond) time difference between the initial and last packets time of a request. Such subtle deltas are not nearly enough to infer the names of all five cities with 95% probability of success based on 300 observations.

Evaluation: The challenge program contains an intended SC Time vulnerability in the *deltas between events in packet exchanges*. The observed data coupled with the restriction in the number of possible minimum spanning trees can be used to create either 4-dimensional Gaussian distributions or a 4-dimensional vector of the mean response times. The minimum Mahalanobis distance between a sample and the set of possible 4-dimensional Gaussian distributions or the Euclidean distance between a sample and the set of possible 4-dimensional vectors can be used to map the response times to a set of the user's query.

Post-Engagement Analysis Ruling: Incorrect

A.3.9.2.5 WithMi 1

A.3.9.2.5.1 Question 010 (SC Space, Intended Vulnerable, Answered Yes)

Response: Vanderbilt responded: *inspection with Wireshark revealed a side channel in space that can be used to determine how many users are involved in a chat that the attacker is not invited to.*

To locate the side channel, we first tried to simulate two interactions -- one with 2 users who were not in a chat together and one with 2 users who were in the same chat. With the expection of forming a chat, the interactions were exactly the same! Inspecting the packets from both sessions with Wireshark showed a sizable package (520 bytes) present in the one where the two users were in a chat that was missing in the other case, meaning that there is some signature associated with forming a chat.

We then experimented with a chat with 3 users. In this case, we found two such sizable packages. This implied to us that a package is sent for each user added to a chat. While we were unable to test this space channel in the presence of more complicated interactions, our experiments implied that there is a side channel in space due to this signature that allows the attacker to determine the number of users within the budget.

The last case to consider is the case where the user is in a chat with only themselves. Since no other user is added to the chat, no signature package can be detected. This means that this case is indistinguishable from the case when there is no chat. In this case, we can use one oracle query to correctly ascertain whether the number of users in the chat is 1 or 0. This can be done within the budget.

Evaluation: Vanderbilt identified the intended SC Space vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.9.2.5.2 Question 053 (SC Space, Intended Not Vulnerable, Answered No)

Response: Vanderbilt responded: w*e do not believe there is a side channel in space that allows an attacker to determine which file was sent. As in the case of question 053, we experimented by*

sending files of the same size and monitoring the traffic with Wireshark. While in question 053, a packet distinctive to the file sent was detectable, no such packet was found in this case. This led us to believe that the application is not vulnerable to a side channel in space that would allow the attacker to determine which file was sent.

Further code inspection confirmed our suspicions. The FileTransfer.send() method included a for loop that randomly sends 0, 1 or 2 extra chunks which destroys the signature of a file.

Evaluation: Vanderbilt accidentally swapped the responses for questions 053 and 051; following confirmation of this mistake, the responses for these questions were corrected. The challenge program does not contain an intended SC Space vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.9.2.6 WithMi 4

A.3.9.2.6.1 Question 027 (SC Time, Intended Vulnerable, Answered Yes)

Response: Vanderbilt responded: y*es, we believe there is a timing side channel that allows an attacker to determine if two users were previously connected. When connecting, the ParticipantConductor.addParticipantToParticipationHistory() method in called. This method checks if the two users connecting have been connected before. If they have not, WithMiConnectionsService.addParticipantToFile() is called which serializes the partipant and writes the serialized code to file. IO operations are expensive and this causes a detectable delay in the time taken to connect. No such serialization and writing occur in the branch taken when the two users have previously connected. This results in a side channel in time which an attacker can use to determine if two u*sers have been previously connected.

Evaluation: Vanderbilt identified the intended SC Time vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.3.9.2.6.2 Question 051 (SC Space, Intended Vulnerable, Answered No)

Response: Vanderbilt responded: *we believe there is a side channel in space that allows the attacker to determine which files a user transferred.*

At first we believed the side channel in space might be due to differing packet sizes based on the size of the file. However, this would not allow the attacker to distinguish between two files with the same size and hence is not strong enough for the question.

We experimented with sending a variety of files of the same size and monitoring the packets sent with Wireshark. This allowed us to see that the packet size associated with sending a given file is distinctive! This is true even for files that are the same size. For example, the files abc.txt, twinkle.txt and pi.txt given in the data/files directory are all 1032 bytes. However, the package sizes associated with them are 303, 269 and 433 bytes respectively. This holds true across multiple runs. Since the attacker is able to see the files in the directory, this would allow the attacker to determine which file was sent.

Evaluation: Vanderbilt accidentally swapped the responses for questions 053 and 051; following confirmation of this mistake, the responses for these questions were corrected. Vanderbilt correctly identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.3.10 Invincea (Control Team)

A.3.10.1 Invincea Overview

Invincea answered 3 questions and had an accuracy of 67%. For these 3 questions Invincea provided 1 exploit which while it exceeded the resource usage limit, also exceeded the input budget.

Table A-61: Engagement 3 Invincea Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	0	0	NA	0	NA
SC Time	0	0	NA	0	NA
SC Space/ Time	0	0	NA	0	NA
AC in Space	1	1	100	1	100
AC in Time	2	1	50	1	50
Total	**3**	**2**	**67**	**2**	**67**

Table A-62: Engagement 3 Invincea Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	2	2	**100**	2	**100**
Not Vulnerable	1	0	**0**	0	**0**
Yes Answer	3	2	**67**	2	**67**
No Answer	0	0	**0**	0	**0**

A.3.10.2 Invincea Specific Responses

A.3.10.2.1 Malware Analyzer

A.3.10.2.1.1 Question 035 (AC Space, Intended Vulnerable, Answered Yes)

Response: Invincea provided an exploit script

Evaluation: The challenge program contained an intended AC Space vulnerability in the packer detector functionality. Given a 256-integer feature vector, while calculating the average value heuristic an attacker can bypass the input guard to trigger an arithmetic exception. This exception is written to the log file. Normal log writes are overwritten, but the exception causes the entire stack trace to be appended to the log file. Invincea identified the same potential vulnerability as Northeastern. The exploit provided by Invincea exceed the resource usage limit of 80 KB.

Post-Engagement Analysis Ruling: Correct

A.3.10.2.2 SmartMail

A.3.10.2.2.1 Question 015 (AC Time, Intended Not Vulnerable, Answered N/A)

Response: Invincea references a writeup which we are unable to locate.

Evaluation: The challenge program does not contain an intended AC Time vulnerability

Post-Engagement Analysis Ruling: None

A.3.10.2.2.2 Question 024 (AC Space, Intended Not Vulnerable, Answered N/A)

Response: Invincea references a writeup which we are unable to locate.

Evaluation: The challenge program does not contain an intended AC Space vulnerability

Post-Engagement Analysis Ruling: None

A.3.10.2.2.3 Question 039 (SC Time/Space, Intended Vulnerable, Answered N/A)

Response: Invincea references a writeup which we are unable to locate.

Evaluation: The challenge program contained an SC Time/Space vulnerability which leaked the administrator of a given mailing list. The vulnerability was in the log file writes that occur when a message is sent. The set of all recipients of a given message (which includes the mailing list administrator and each of the subscribers) is alphabetized and written to a log file with the mailing list subscribers written to one log file and the administrator written to another. Using this side channel an attacker can run a binary search for each character and determine each subsequent character of the administrator's email beginning at the first character.

Post-Engagement Analysis Ruling: None

A.3.10.2.3 Tour Planner

A.3.10.2.3.1 Question 008 (AC Time, Intended Not Vulnerable, Answered Yes)

Response: Invincea provided an exploit.

Evaluation: The challenge program does not contain an intended AC Time vulnerability. Invincea created an exploit which submits a route request between a large set of points using longitude and latitude references. The resulting tour takes 61.14 seconds to complete. The challenge question limits the user input to 500 bytes and Invincea's exploit requires 15503 bytes of input. This potential vulnerability is due to a lack of enforcement of the number of places a user can submit a tour request between.

Post-Engagement Analysis Ruling: Incorrect

A.3.10.2.4 Tweeter

A.3.10.2.4.1 Question 049 (AC Time, Intended Vulnerable, Answered Yes)

Response: Invincea responded: w*e believe that an algorithmic complexity vulnerability in time exists. The vulnerability is spread across a number of methods in a number of classes, but generally has to spell checking functionality of a tweet.*

SpellingCorrectionService.Task.run():

This method breaks up a tweet into words where the delimeters for words are anything other than the characters @,#,a-z,A-Z. Each word that does not begin with an @ or # is passed to the Spelling.correct() method.

Spelling.correct():
This method first checks that the length of the word is <= MAX_WORD_LENGTH + 2.
By parsing the dictionary.txt file contained in the jar we were able to determine that MAX_WORD_LENGTH = 24, so the actual max length of a word can be 26 characters.
Next, the word is passed to the isPossible() method to determine if any possible corrections to the word exist. If so, then the word is passed to the getCorrections() method to actually generate corrections. Interestingly, these two methods both work by calling the same set of functions to generate cadidate words that are then checked to determine of the words are valid. The differences in call-chains is illustrated below.
isPossible()->generateKEdits()->generateP1Edits()->generate1Edits()
getCorrections()->generateP1Edits()->generate1Edits()

The candidate words are generated in generate1Edits() and then checked in generateP1Edits() by searching a trie that was generated from dictionary.txt.

Spelling.generate1Edits():
Here are the transformations that take place on the word in this function. In all cases we refer to the length of the original word as n
1) Generate n candidates by removing a character at each index. Each new candidate will have length n-1
2) Generate n-1 candidates by swapping adjacent characters
*3) For each index in the word generate 26 new candidates by replacing that index with a-z. This results in 26*n candidates*
*4) For each index in the word generate 26 new candidates by inserting the letters a-z at that index. This results in 26*n candidates*
The total number of candidates generated for a word is: 54n-1

PoC:
In the worst case doing a lookup in a Trie is O(m) where m is the length of a search string.
If we are able to construct input strings that never match all the way to a leaf node, and whose generated candidates will also never quite match, then we will do a worst-case lookups on all input strings and their candidates. This is where the algorithmic complexity vulnerability is really exploited. Unfortunately, we did not have enough time to figure out what the perfect string would be to do this.
Instead we decided to generate random words of length 26 (the max size we know the program accepts accepted) and concatenate them together (separated by spaces) into one tweet whose length is within the input budget. Doing so will allowed us to pass the resource utilization required. However, because the tweet itself was at the limit of the input budget we realize that the addition of HTTP headers put us over the input budget. So while it might not be a that fits within the input budget, it provides enough confidence that with more deliberately crafted strings a solution that fits within the budget is likely to exist.
The following is the command-line used to generate random tweets composed of 80 26 character words (note that the interact.sh script from the examples directory is being used):

interact.sh sendTweet user password "$(cat /dev/urandom | tr -dc 'a-z' | fold -w 26 | head -n 80 | tr '\n' ' ')"
The specific random data that caused over 25 minutes of processing, along with the curl command that sent it is as follows. Note, you can't cut and paste this because the csrf token won't be valid. Instead, cut and past the tweet text itself.

Evaluation: The challenge program contained an AC Time vulnerability in the spell-correcting feature. The spell check algorithm's runtime is a function of word length and number of mistakes; however, the application has a maximum for word length and for number of mistakes. By providing a word which is within the applications maximum word limit (26 characters) and contains just enough mistakes to trigger the spell check algorithms slow execution time, the resource usage limit can be exceeded. Invincea flagged the spell correcting feature as vulnerable.

Post-Engagement Analysis Ruling: Correct

A.4 Engagement 4

The overall accuracy of the Blue Teams was 70% (66% when not including the control team). Of the 268 questions answered, 155 were answered "yes" and 112 were answered "no". In comparing E4 and E3 to E2 and E1, R&D Teams collectively answered a greater number of questions with improved accuracy. The total accuracy for R&D Teams increased by 7 percentage points with R&D Teams more than doubling their total number of responses from 114 in E3 to 229 in E4. This is a similar pattern to what occurred from E1 to E2 where R&D Teams' combined accuracy increased by 15 percentage points to 78 percent with teams more than doubling their total responses from 65 in E1 to 133 in E2.

A.4.1 University of Colorado Boulder

A.4.1.1 UC Boulder Overview

UC Boulder answered 27 questions with an 85% accuracy. They had the highest accuracy (100%) for SC Space questions. Their accuracy increased 6 percentage points from their E2 accuracy of 79% with the team answering 2 fewer questions. Their SC Space and AC Space accuracies increased by 25 and 48 percentage points respectively from E2; however, their SC Time and AC Time accuracy decreased by 36 and 14 percentage points. The team answered 3 more SC Time questions and 5 fewer AC Time questions in E4 than in E2. In E4, Colorado had the highest TNR against the set of answered questions.

Table A-63: Engagement 4 UC Boulder Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	8	100	8	100
SC Time	4	2	50	2	50
SC Space/ Time	0	0	NA	0	NA
AC in Space	8	7	88	7	88
AC in Time	7	6	86	6	86
Total	**27**	**23**	**85**	**23**	**85**

Table A-64: Engagement 4 UC Boulder Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	21	17	**81**	17	**81**
Not Vulnerable	6	6	**100**	6	**100**
Yes Answer	17	17	**100**	17	**100**
No Answer	10	6	**60**	6	**60**

A.4.1.2 UC Boulder Specific Responses

A.4.1.2.1 Airplan 1

A.4.1.2.1.1 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Response: "[…] The vulnerability reported by the Auditr tool is based on the \crew management request. The attacker first uploads the malicious route map and then requests the minimum number of crews for the map. The total PDU size of these requests is 11,902 bytes which is well below the budget of 25,000 bytes. Furthermore, this request causes the application to take 960 seconds, which exceeds the specified resource usage limit of 500 seconds. The vulnerability found by fuzzing causes an infinite loop due to a negative weight cycle in the graph. Exploit A (Auditr): This exploits an underlying vulnerability in the \Find the Minimum Number of Crews functionality that is computationally expensive for certain route map types. The root cause of this inefficiency is in the Limit.limit method, which works over a slightly modified version of the graph that contains some additional nodes and vertices. This method triggers an inter-procedural loop, between methods Limit.limit and Limit.search. Additionally, the inner loop reaches the method Shifter.changingArrange, which is a recursive implementation of merge sort. An attacker can control the loop bounds for both loops by uploading a malicious route map. The malicious map contains a relatively small cycle of airports where all nodes in the cycle contain multiple identical edges. This input causes the merge sort method to be called a quadratic number of times. Our toolchain was quite useful for finding this vulnerability. Since the question asks about the presence of an algorithmic complexity vulnerability, we first used Auditr to find loops and recursive procedures whose termination condition is input-tainted. Since many procedures in the application are called using reflection and therefore appear unreachable to the static analyzer, we first modified the main method of the application to explicitly call methods that handle HTTP requests. With this modification, Auditr reported 14 loops/recursive procedures whose termination condition is controlled by the attacker. Furthermore, we noticed that the recursive Shifter.changingArrange procedure was called from a user-controlled loop found in the Limit.search method. […] Given this information, we manually constructed an exploit that traverses this code path and causes a vulnerability. […]"

Evaluation: Colorado identified an unintended vulnerability in the crew scheduling implementation of the challenge program.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.1.2 Question 051 (AC Space, Unintended Vulnerable, Answered Yes)

Response: "The shortest path algorithm, implemented in method OptimalPath.calculateOptimalPath, maintains a priority queue while traversing the graph's

airports. This queue contains all the airports that the algorithm has already visited along with their current cost. At every step of the algorithm an airport is added to the queue either if it is not already in the queue or if it is discovered with a lower cost. The new cost is calculated by using a heuristic method, implemented in PathHeuristic.heuristic. The vulnerability in the challenge program stems from the fact that the priority queue has an unbounded size. Specifically, if an attacker uploads a map with a negative cycle (i.e., with a negative sum of weights) and every node in the cycle has some neighbor nodes that cannot reach the target node, this causes the priority queue to grow in every iteration of the main algorithm because these nodes are being discovered with an infinitely decreasing cost. […]"

Evaluation: Colorado identified an unintended vulnerability in the shortest path implementation.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.2 Airplan 2

A.4.1.2.2.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Response: Colorado repeated the analysis performed for Airplan 1 question 020.

Evaluation: The challenge contains an intended vulnerability in the sort implementation. If the number of items to be sorted is divisible by 6, the slow path is triggered. Colorado identified an unintended vulnerability in the crew scheduling implementation of the challenge program.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.2.2 Question 038 (SC Space, Intended Vulnerable, Answered Yes)

Response: "[...] Themis leverages a taint analyzer as the front-end to identify a set of hotspots that need to be analyzed more precisely using the QCHL verifier, and each hotspot is a method of which arguments are tainted by the secret sources. Since the verifier fails to prove that current program obeys the non-interference property, the program may be vulnerable to a potential side-channel attack. Although automatically synthesizing an attack vector is beyond the scope of the tool, we were able to construct an attack guided by the hints provided by Themis. Specifically, since the operational budget only allows 2 operations, one of these must be an active operation where the attacker issue a single application request and monitors the corresponding response traffic. The query passenger capacity matrix will return a matrix with the passenger capacity between airports with the origin airports on the left and the destinations on the top. The class FlightMatrixManager is responsible for constructing the passenger capacity matrix and building the corresponding HttpManagerResponse. Each cell in this matrix is formatted to have exactly 10 characters. The id of each airport can have at most 3 characters and the passenger capacity is represented by an int, hence 10 characters is guaranteed to have enough space to represent both the id of the airport as well as the capacity between airports. Since each cell has always 10 characters, we can model the response to the passenger capacity matrix query by an injective quadratic function $f(x) = a * x^2 + b * x + c$, where x is the number of airports and $f(x)$ is the observed traffic. This function is injective since two routes r_1 and r_2 will have the same response size $f(x)$ if and only if they have the same number of airports. If the attacker is able to construct such function, then she can it use to determine the number of airports in a user's stored route map. [...]"

Evaluation: Colorado identified the intended vulnerability, but there appears to be some confusion on the operational budget. The operational budget provides two operations, while

active operations are an option they are not required to exploit this vulnerability. The attacker observes the user's request (1 operation) and uses a single oracle query to confirm the secret (1 operation). For STAC, it is assumed that the attacker has a copy of the application code. Any analysis the attacker performs on the copy (e.g. identifying the relationship between number of airports and packet size) are not counted as part of the operational budget.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.3 Airplan 3

A.4.1.2.3.1 Question 002 (SC Space, Intended Vulnerable, Answered Yes)

Response: "We found this vulnerability with the help of our Themis tool, a static analyzer based on Quantitative Cartesian Hoare Logic (QCHL) […]. This output indicates that the two methods getContentsForPost and handlePost were deemed potentially vulnerable using the less precise taint analyzer and therefore were analyzed using the QCHL relational reasoning engine of Themis. [...] The class MapPropertiesCoach is responsible for retrieving the map properties and building the corresponding HttpCoachResponse. Each property field will be formated using the String format(String value) method in the MapPropertiesCoach class. This formatter increases the padding of each field if its length is larger or equal to $valueLength$ and updates the value of $valueLength$ to $valueLength * 2$ ($valueLength$ is initialized to 19). The attacker's goal is to retrieve information regarding the connectivity field which can be either Fully Connected (15 characters) or Not Fully Connected (19 characters). If the route is not fully connected then the padding of each field will be updated to 38 characters and all 10 properties will have this padding. The attacker can use this information to determine if a user's route is strongly connected or not."

Evaluation: Colorado identified the intended vulnerability. See comment on Airplan 2 question 038 concerning the operational budget.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.3.2 Question 016 (SC Space, Intended Not Vulnerable, Answered No)

Response: "We verified the absence of response-size-side-channel with the help of our Themis tool, a static analyzer based on Quantitative Cartesian Hoare Logic (QCHL) for verifying the bounded non-interference property. [...] To further understand why the program is safe, we also performed a comprehensive study on methods that are related to generating the response. Specifically, we first tried the same attack as in Question 38 presented in Subsection 2.3.2 but were not able to derive a function $f(x)$ that models the response size of the passenger capacity matrix. Airplan 3 has a class CellFormatter that randomizes the amount of padding for each cell of the matrix. Since the padding for each cell is random, the function $f(x)$ is no longer injective, i.e. for the same x value (number of airports) we observe different response sizes. For example, for a route with 1 airport we performed 5 passenger capacity matrix queries and observed response sizes 3324, 3366, 3239, 3300 and 3324. Since the response size can vary for the same input, we are not able to construct an injective function $f(x)$ that would allow us to attack this variant of Airplan. […]"

Evaluation: Colorado correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.3.3 Question 024 (AC Time, Intended Vulnerable, Answered Yes)

Response: Colorado provided the same response as for Airplan 1 question 020 and Airplan 2 question 037.

Evaluation: The application contains a vulnerable Ford-Fulkerson implementation is crew scheduling. Colorado identified an unintended vulnerability in the crew scheduling implementation of the challenge program.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.4 Airplan 4

A.4.1.2.4.1 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

Response: Colorado provided the same response as for Airplan 1 question 020, Airplan 2 question 037, and Airplan 3 question 024.

Evaluation: The applications contains a vulnerable Dijkstra's implementation. Colorado identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.4.2 Question 055 (SC Space, Intended Not Vulnerable, Answered No)

Response: Colorado repeated the analysis from Airplan 3 question 002 and concluded that the application was not vulnerable.

Evaluation: Colorado correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.5 Airplan 5

A.4.1.2.5.1 Question 005 (SC Space, Intended Not Vulnerable, Answered No)

Response: Colorado repeated the analysis for Airplan 2 question 038 and concluded that the application was not vulnerable.

Evaluation: Colorado correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.5.2 Question 052 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Yes, there is an algorithmic complexity vulnerability in space that causes the challenge program's memory usage to exceed the resource usage limit. The directory exploit 052 contains scripts that will trigger the uncovered algorithmic complexity vulnerability in space. Our exploit uses a total PDU size of 2,269 bytes, and causes the space usage of the application to exceed the specified 1024 MB. The exploit is similar to the one described in the answer of Question 51. However, unlike in Question 51, we can only exploit the underlying vulnerability by creating an input map with a negative-weight cycle where neighbors of some nodes in the cycle have a strictly positive out-degree. The root cause of the vulnerability is that the ShortestTrail.calculateShortestTrail method does not properly handle negative-weight cycles. This method maintains a priority queue of airports visited by the algoritm, along with their current cost. An airport is added to the priority queue if it is not already in the queue or if it is visited using a path with lower cost. However, the algorithm allows infinitely many elements to

be added to the priority queue, causing the memory usage of the priority queue to grow indefinitely. […]"

Evaluation: The challenge contains a zip bomb-style vulnerability in the parsing of submitted route maps. Colorado identified an unintended vulnerability in the handling of negative-weight cycles.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.6 BidPal 1

A.4.1.2.6.1 Question 033 (AC Time, Unintended Vulnerable, Answered No)

Response: "No, there is no vulnerability in time that would cause the challenge program to exceed the resource usage limit. Our justification is based on CodeHawk analysis and manual code inspection. CodeHawk was used to produce (a) the list of intraprocedural loops whose termination condition depends on user input, and (b) the call graph. We used CodeHawk to produce a list of intraprocedural loops whose termination condition depends on user input. Figure (1) below shows a part of the CodeHawk output that lists the methods with loops and user input taint. We analyzed these methods manually for potential vulnerabilities based on various possible values of user input. For some of these loops, the loop termination was determined by a randomly generated number (for example in the case of edu.computerapex.buyOp.messagedata.ExchangeData.make) or a variable that is not input-a ected. No vulnerabilities were found in any of the methods. We used CodeHawk also to produce a call graph of the program. We then manually inspected the call graph to find the interprocedural loops in the program. All the loops were determined to be safe. The challenging case was that of the interprocedural loop that causes the time vulnerability in bidpal 2. [...]"

Evaluation: Draper, Vanderbilt, and Invincea identified unintended vulnerabilities in the application.

Post-Engagement Analysis Ruling: Incorrect

A.4.1.2.7 BidPal 2

A.4.1.2.7.1 Question 044 (AC Time, Intended Vulnerable, Answered Yes)

Response: "Yes, there is a vulnerability in time that would cause the run time of the challenge program to exceed the resource usage limit. This vulnerability is based on an interprocedural loop between the org.techpoint.sale.AuctionDirector.sendExchangeData and org.techpoint.sale.AuctionDirector.auditAndSendExchangeData methods. To discover the vulnerability (and exploit), we used the CodeHawk tool to produce (a) the list of intraprocedural loops whose termination condition depends on user input, and (b) the call graph. After manual inspection, we identified an interprocedural loop as the most likely to be vulnerable. The reverse call graph generated using CodeHawk in Figure (4) shows the interprocedural loop between the two methods org.techpoint.sale.AuctionDirector.sendExchangeData and org.techpoint.sale.AuctionDirector.auditAndSendExchangeData. We then analyzed this loop in detail and found the vulnerability. The vulnerable method is org.techpoint.sale.AuctionDirector.verifyData that is called when serializing a BidComparisonData object for transmission to another user for bid comparison. This method is called by the org.techpoint.sale.AuctionDirector.auditAndSendExchangeData method, which itself is called by org.techpoint.sale.AuctionDirector.sendExchangeData. Hence these two

methods are part of an interprocedural loop – that is, org.techpoint.sale.AuctionDirector.sendExchangeData and org.techpoint.sale.AuctionDirector.auditAndSendExchangeData call each other in a loop or exit the loop based on whether org.techpoint.sale.AuctionDirector.verifyData returns false or true respectively. We can make this method return false based on a very specific input and hence make the application get stuck in an infinite interprocedural loop. […]"

Evaluation: Colorado identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.7.2 Question 057 (SC Time, Intended Vulnerable, Answered No)

Response: "No, there is no side channel in time that would allow the attacker to determine the bid of another user to within $30 accuracy. We get evidence for this from the Discriminer tool. The results of the tool tell us that it is not the case that one can distinguish the time of the secret-to-the-attacker bids with the 7 following values (at intervals of 30$ - 0, 30, 60, 90, 120, 150 and 180). The tool examines the data set where the attacker bid is varied (0 to 500), and the secret-to-the-attacker bid is chosen from among these seven values. If, as the attacker bid changes, the running time of the program for these 7 secret-to-the-attacker bid values could be differentiated into clusters (with one cluster corresponding to each secret bid), the Discriminer tool would discover these clusters with good accuracy. We did this experiment, and the accuracy was only 28%, providing evidence there is no time side channel. For illustration: the actual plot for the 7 victim bid amounts mentioned above (for the attacker bid varying from 0 to 500) versus the time measured by the attacker is shown in Figure (6). Colors correspond to different secret values. As can be seen, there are no obvious clusters and the timing observations are essentially random and have no meaningful correspondence to the bid amount of the victim. We provide further evidence for the absence of the time side channel by manual code inspection of the critical part – the bid comparison. A brief description of the process of comparison of bids by two users is given and then it is explained why there is no side channel in time. The BidComparisonData object that user A sends to user B to compare bid amounts holds an array such that $array[i] = [x^e \ mod(n) - bidAmountofB \ + \ i] \ (mod \ p)$. The $bidAmountofB$ is a secret to A since the value of $[x^e(mod \ n) - bidAmountofB]$ is grabbed as a whole from B's BidCommitmentData object and x is a random integer created by B. A embeds his bid information into the array by adding a certain large value $(mod \ p)$ to all array entries starting from $array[j]$ where j is equal to A's bid, hence effectively scrambling all array entries with index greater than or equal to A's bid. B determines whether his bid is greater/less than A's bid by checking whether $array[bidAmountofB] \ == \ x$ in the received BidComparisonData object. If $array[bidAmountofB] \ == \ x$, then it means that B's bid must be at least be equal to A's bid, since the value of x at $array[bidAmountofB]$ was scrambled by A. In the BidComparisonData object created as explained above, the bid amount does not affect the timing but only affects the length of the array. This is because during the serialization of the BidComparisonData object (more specifically during the process of adding the array entries to the byte array), the process is randomized in time in org.techpoint.sale.messagedata.AuctionProtoSerializer.serialize method by using Math.random() in the for loop that iterates through all array entries and adds them to the byte array. Hence, even though it could be possible to guess the bid using the length of each entries in the array, it is not possible to do so using a timing analysis. Since observing the time it takes to serialize the BidComparisonData object leaks no information about the contents of the

array that would enable the attacker to determine the victim's bid amount, we conclude that the application is safe from side channels attack in time."

Evaluation: Colorado did not identify the intended vulnerability, that the bid comparison process contains a loop that depends on the bid value. The time for the application to perform the bid comparison leaks the user's secret bid.

Post-Engagement Analysis Ruling: Incorrect

A.4.1.2.8 Info Trader

A.4.1.2.8.1 Question 008 (AC Time, Intended Not Vulnerable, Answered None)

Response: "Our fuzzer produced multiple inputs that can crash the server and this may be exploitable. Yet we don't expect that an input that crashes the server is an intended vulnerability. The attached python script info trader fuzzer.py performs a systematic fuzzing by creating random documents and adding them. Each document links to a randomly selected list of other documents including itself. When a document is created that links to itself, it causes an infinite loop. A typical fuzzed document is given in the files exampledoc2.txt and exampledoc3.txt. Based on this, we modified the given example as exampledocument.txt. To reproduce the vulnerability, run (after starting the server): ./postdocument.sh. After about 3 minutes, we obtain a crash due to an infinite recursion that causes a stack overflow"

Evaluation: No official response provided. The stack overflow exception is purposefully placed to prevent an AC Time vulnerability but to appear as though it might be vulnerable.

Post-Engagement Analysis Ruling: NA

A.4.1.2.9 Linear Algebra Platform

*A.4.1.2.9.1 Question 050 (AC Time, Intended Vulnerable, Answered *Yes)*

Response: "We used structured fuzzing to look into exploit for this problem. Our fuzzer quickly reported some 6x6 matrices for which the server was taking a lot of time to respond. Upon further investigation, we found that in the matrix multiplication operation, the task of multiplying two matrices is divided into two threads based on the skewness of one of the matrices. If the input matrix is designed in such a way that the skewness is greater than equal to $ln(2)$, then the entire application is handled by one thread and the other thread is idle. Due to a concurrency vulnerability in the implementation, the application then deadlocks indefinitely. This bug is well within the resource budget as the input matrices are only 6x6. Yet we don't expect that an input that deadlocks the server is an intended vulnerability."

Evaluation: Colorado identified the vulnerable behavior of the matrix multiplication implementation resulting in the ability to have all the computations performed on a single thread. It appears from the provided response that during testing, they observed a locking scenario on the intended vulnerability which may have caused the teams to flag this as an unintended vulnerability and chose not to provide a response. In either case, the team identified the intended vulnerability of the application and the details resulting in all the computations being handled by a single thread. As a result, we have changed the response provided to "Yes" and provided a post engagement analysis ruling of correct.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.10 Malware Analyzer

A.4.1.2.10.1 Question 023 (AC Space, Intended Vulnerable, Answered No)

Response: "No, there is no algorithmic complexity vulnerability in time that would cause the real runtime of the challenge program to exceed the resource usage. Here, for each request from the user, the system will store the content of the response in a log file (response.log). Therefore, the logical size of response.log is equal to the response size. Although our static analysis tool Themis is not able to directly prove the absence of this complexity vulnerability, we were able to use its output to identify a subset of relevant procedures and perform a case study for each request in Figure 7. […] the complexity in space when there is an exception during the request. Each cell that is marked as *constant* represents a bounded constant string from the server […] None of these constants exceeds the indicated resource usage limit of 80K.

The cells marked as *bounded* […] indicate that the response size is strictly bounded. Since the corresponding request contains at most five records and the size of each record is a very small constant (an MD5 string together with a double), the total response size cannot exceed 80K. In the case of request list dasms, the response size will reach 80K only if current database has at least 2560 dasms. However, assuming each dasms has an MD5 of 32 bytes, current input budget (3,000 bytes) cannot generate 2560 dasms. Therefore, this is not a worst-case exploit […] the response size of get cfg is bounded by twice the maximum dasm uploaded by the user using another request add dasm. Here, the functionality of get cfg is to print the control flow graph using an MD5 string. However, while the user uploads a dasm using request add dasm, the method make function cfg() maintains a strong invariant to make sure that each basic block has at most two successors (using a conditional jump). As a result, suppose each assembly instruction uploaded by the user represents is a basic block and each of them has two successors, then the output of the control flow graph will be at most twice the size of the original input. Since the input size is at most 3,000 bytes, the response size will not exceed 80K. To conclude, we believe there is no algorithmic complexity vulnerability in time that would cause the real runtime of the challenge program to exceed the resource usage."

Evaluation: Colorado did not identify the intended vulnerability, that an integer overflow can be used to trigger a divide by zero exception. The entire stack trace of this exception is then written to file.

Post-Engagement Analysis Ruling: Incorrect

A.4.1.2.11 PowerBroker 1

A.4.1.2.11.1 Question 019 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Yes, there is an algorithmic complexity vulnerability in space that will cause the challenge program to store a file with logical size larger than 1,000,000 bytes. The application limits the maximum number of auctions each user can perform. This is done in the method void sendAnnounceMessages() in the class AuctionAnnounceStage, where the number of auction is bounded by 5: if (numAuctionsSoFar < 5). However, since this is a peer-to-peer application, the attacker can create a rogue client by changing the byte code of her client and modifying the limit on the number of auctions to a large number, e.g. 100,000. The attacker can then use a profile to sell a very large number of auctions (e.g. 10,000) which will increase the size of the local file \stac.log.1" to a value larger than 1,000,000 bytes. This attack is possible since the sellers are the

only ones that check the maximum number of auctions that can be performed and the buyers will just bid on any available auction.

We can use CodeHawk to examine the path in in the client between receiving a message that an auction is starting and the logging of the message. We can check that there are no safeguards along this path. Similarly, we can use CodeHawk to analyze the routes in the rogue client between creating and sending auctions to the victim clients. There is such a path between sendAnnounceMessages and ProtocolsConnection.write and, and by modifying the guard in this method we can achieve an exploit."

Evaluation: The challenge contains an intended vulnerability in the connection establishment. An attacker can start a connection and terminate before the connection is established. The victim user continuously attempts to re-connect, logging each failed attempt to disk. Colorado identified an unintended vulnerability in that the check on the number of auctions is only performed on the *sender* side (by the seller).

Post-Engagement Analysis Ruling: Correct

A.4.1.2.12 PowerBroker 2

A.4.1.2.12.1 Question 025 (AC Space, Unintended Vulnerable, Answered Yes)

Response: Colorado repeated the analysis performed in PowerBroker 1 question 019.

Evaluation: The challenge does not contain an intended vulnerability. Colorado identified an unintended vulnerability in that the check on the number of auctions is only performed on the *sender* side (by the seller).

Post-Engagement Analysis Ruling: Correct

A.4.1.2.13 Tour Planner

A.4.1.2.13.1 Question 011 (SC Space, Intended Not Vulnerable, Answered No)

Response: "No, there is no side channel in space from which a third party can discover which cities the user would like to visit (unordered list). We verified the absence of response-size-side-channel with the help of our Themis tool, a static analyzer based on Quantitative Cartesian Hoare Logic (QCHL) […] Themis believes that the program obeys the ϵ-bounded non-interference property, indicating the absence of space-related side channels. To further understand why the program is safe, we also perform a study in which we model the problem using subset sum. Specifically, we assume that the attacker knows the 25 possible cities since he can retrieve this information by sending a *places* query to the server. When the user sends a query to the server with a list of cities, the attacker can observe the response traffic and conclude that the size of the response grows linearly with the number of cities. However, for any possible combination of 5 cities, it cannot determine with 99% probability and 3000 operations which cities are part of the user's route. […]"

Evaluation: Colorado correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.13.2 Question 045 (SC Time, Intended Vulnerable, Answered No)

Response: "We verified the absence of timing-channel with the help of our Themis tool, a static analyzer based on Quantitative Cartesian Hoare Logic [...] Themis was able to verify the desired

property. To further validate the output produced by Themis, we perform dynamic analysis and observe the running time of random samples. Specifically, we observed the running time of the 907 possibilities when the response has size 609, which corresponds to an input from the user using 5 cities when the sum of their characters is equal to 36 (counting the initial city twice). Even though this is just a small subset of the possible inputs, we will show that even for this small subset we cannot differentiate between the running times of more than 300 inputs, and therefore cannot infer which cities were chosen by the user with 300 operations and with a probability of success of 95%. For each input, we measure 10 independent runs in the reference machine and computed the average time for each run.

Figure 12 shows the distribution of the running time in milliseconds. The several runs are groups in small intervals of 1.67 milliseconds. If we were able to have this precision, then it would be possible to distinguish these subset of the inputs within 300 operations. However, the running time distribution of the observations has an average of 276.598 seconds with a standard deviation of 3.616 seconds. If we count the number of observations that are in the interval $[276.598 \pm 3.616]$, we can see that there are 553 out of 907 observations that fall within this interval. This makes it impossible to distinguish these observations from each other with 300 operations and 95% of success. Note that the observed 907 queries are only a small part of the entire search space that has 201,593 possibilities. Note that the attacker could measure time in microseconds by using a tool such as tcpdump and measuring the time between the request of the user and the response of the server. However, even with measurements in microseconds, the time distribution will follow a similar pattern to the one shown in Figure 12 and it would not allow the attacker to distinguish between all 201,593 possible observations for 5 cities into 300 or less clusters with a 95% confidence."

Evaluation: Colorado did not identify the intended vulnerability, that the server provides a set of response times for each user request. This set of response times can be used to map to a user request (see Tour Planner whitepaper for more information). Colorado's response indicates that they performed analysis on the total response time from request to response and did not analyze the response times between subsequent packets. Additionally, the team states that there are 201,593 possible observations for 5 cities, but the side channel is invariant to the order of the cities, reducing the set of possibilities to 53,130.

Post-Engagement Analysis Ruling: Incorrect

A.4.1.2.14 Tweeter

A.4.1.2.14.1 Question 021 (AC Space, Intended Vulnerable, Answered Yes)

Response: "The maximum permitted size for a tweet is 140 characters. However, there is a vulnerability in Tweeter that allows a user to post of tweet of size up to 250 characters. Using quick fuzzing we noticed that tweets that contain at least two hastags (#vulnerability #memory) result in an increase in the size of the tweet posted to the server. We used cma-es guided search to explore the space of up to 140 characters strings made of hashtags and a subset of lower and upper-case letters to find tweets such as #A#A#A#A#A#A#A#A#A#A#A#A#A#A#A that get posted as a 241-character string. Interestingly, for the tweet of larger size with the similar pattern, the server throws the following exception: could not execute statement; SQL [n/a]; nested exception is org.hibernate.exception.DataException: could not execute statement. This vulnerability permits a way to quickly rampup database size. […]"

Evaluation: The challenge contains an intended vulnerability in that users can directly specify the size of the profile image. An attacker can use the budget to create multiple user accounts and set the largest possible size for each profile image to exceed the resource usage limit. Colorado identified an unintended vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.15 WithMi 2

A.4.1.2.15.1 Question 012 (SC Space, Intended Not Vulnerable, Answered No)

Response: "The application is not vulnerable to space side-channel vulnerability. The application uses Huffman coding to compress files. Unlike the normal Huffman coding in which the most frequent characters are inserted near to root node, this implementation inserts least frequent characters at the top of tree near to root node and most frequent characters at the bottom near to leaf nodes. If we analyze the five files: abc.txt, ipsum.txt, pi.txt, twinkle.txt, and twinkle2.txt, we see that the frequency of characters in abc.txt and pi.txt is similar for most of characters. In other words, characters are almost uniformly distributed in abc.txt and ipsum.txt. This indicates that the compression can work without considering the order of inserting least or most significant characters. However, the frequency of characters for other files are not uniformly distributed, we have characters with very low frequency and characters with high frequency. This means that the compression can cause the size of files to increase since it inserts least significant characters at the top of tree which requires lower number of bits to encode, but the most significant characters at the bottom of tree which requires higher number of bits to encode. This observation indicates potential space side-channel since files get different sizes after compression and we can guess which files are shorter or bigger after processing. Even though the files are different in size after compression, the application sends the files in packets with the same size. In addition, the application adds random bytes to the last packet. As a result, it is not possible to distinguish all files in two operations. The only situation in which we can distinguish files with two operations is the case when the application sends files in 4 packets (three with 256 Byte chunks before packet headers and the last one with less than 256 plus random bytes). As the last one contains random bytes, we can guess that the file is either abc.txt or pi.txt since they can be sent with three 256 Bytes packets. However, in all other cases, it is not possible to guess file name in two operations since all of them are randomly sent in either 5 or 6 packets where all packets are the same size except the last one. However, the last one includes random bytes that prevents any observation through space size. As a result, it is not possible to distinguish them in 2 operations with 99% success."

Evaluation: Colorado correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.16 WithMi 4

A.4.1.2.16.1 Question 010 (SC Space, Intended Vulnerable, Answered Yes)

Response: "Yes, withmi 4 is vulnerable to a side-channel attack in space from which an attacker can determine which file was transferred from the victim to the participants in a chat that the attacker is not part of. The vulnerability is in the method com.digitalpoint.togethertalk.FileTransfer.send this method fails to either equalize or randomize the sizes of the packets that transfer the compressed contents of the file, allowing an attacker with access to the set of candidate files to be transferred (as assumed in this question), to create

`size patterns' for each file and to monitor the stream of packets originating from the victim for the occurrence of these patterns. […]"

Evaluation: Colorado identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.16.2 Question 031 (SC Time, Intended Vulnerable, Answered Yes)

Response: "Yes, there exists a side channel in time from which the attacker can determine whether two users have been previously connected. The vulnerability is in the method com.digitalpoint.togethertalk.MemberManager.addMemberToMemberHistory The variable previouslyConnected indicates whether two users have been previously connected. When this variable is true the code path through pc=66 (see the control flow graph for this method in Figure 15), is taken, which has a significantly higher cost than the path through pc=83, which is taken when two users have not been previously connected. For a side channel to exist, however, there must be both a dominating and a post-dominating external action by which this discrepancy can be externally observed. The post-dominating action is provided by the call to this.WithMi.transmitReceipt(1,..) at pc=141 (in the last basic block), which results in a message being sent to the peer with whom the connection was established, which can be observed by the attacker. […]"

Evaluation: Colorado identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.16.3 Question 036 (AC Space, Intended Vulnerable, Answered Yes)

Response: "The vulnerability is in the decompression method com.digitalpoint.smashing.SneakerWrapper.stretch. This method reads, within a loop, a character, b from an input stream and then reads the following 4 bytes as an integer, num. It then writes the character b num times to the output stream. This loop is repeated until no more (b, num) pairs are available from the inputstream. During this decompression, the total size of the decompressed output is maintained and an exception is thrown if this total size exceeds 100,000,000 (see Figure 19). That is, this decompression method allows a 5-byte input stream to be expanded into a 100,000,000 bytes output stream. The SneakerWrapper.stretch method is called from the dispatcher method that receives the messages (see Figure 20); one call is made for each file transfer message received. Thus, to output a file of 120MB requires the attacker to send just two messages with file content ('a', 100,000,000) and ('a', 2582912) (represented in byte format). The messages themselves are somewhat larger than 5 bytes, as they include identity information for authentication and total size and offset, but these two messages are still well within the budget of 10,000 bytes. One slight complication is that the withmi 4 client as is will not generate these messages. The method responsible for sending files, com.digitalpoint.togethertalk.FileTransfer.send breaks up the input file into chunks of 256 bytes, which would result in the creation of 491520 chunks, and thus 491520 file transfer messages to be sent for a 120MB file, well exceeding the input budget [resource usage limit]."

Evaluation: The challenge contains an intended vulnerability due to an integer overflow in the RLE compression. Colorado identified an unintended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.17 WithMi 5

A.4.1.2.17.1 Question 014 (AC Space, Intended Not Vulnerable, Answered Yes)

Response: Colorado reported the same vulnerability as in WithMi 4 question 036

Evaluation: Colorado identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.1.2.17.2 Question 027 (SC Time, Intended Not Vulnerable, Answered No)

Response: "To establish that no information regarding having been previously connected is leaked via timing during the process of connecting, it suffices to show that there is no significant difference in time spent within the handshake sequence that takes place during connecting, as subsequent packets would be created by user commands and hence their timing cannot carry any information about the state of the system during connecting. The handshake sequence consists of four messages sent from client to server with the corresponding four messages sent from server to client in reply, following a state machine implemented in DialogsCryptoState. The first four messages from the client and the first three replies from the server are sent from within the code for this class. The DialogsCryptoState object is created for the channel on which the connection is received; it has no access to the state of the application itself (com.techtip.chatbox.DropBy) or its customerManager (com.techtip.chatbox.CustomerManager), which is the only object that keeps a list of past connections. Hence the code in DialogsCryptoState cannot take different code paths based on the presence or absence of previous users, and therefore the timings of the first four messages from the client and the first three messages back from the server cannot be affected this presence or absence either. Thus, it remains to be established that the timing of the reply by the server to the fourth client message in the handshake is not significantly affected by whether the users have been previously connected. […] The cost model indicates that the timing cost differences between the alternative paths is very small, and therefore unlikely to be observable. Based on this static analysis and theoretical examination we conclude that there is no side channel in time by which the attacker can deduce whether two users have been previously connected. We confirm this conclusion by running some experiments for the two alternatives and examining the timings of the handshake sequences. […] Clearly there is no discernible separation between the two cases."

Evaluation: Colorado correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.2 Draper Labs

A.4.2.1 Draper Labs Overview

Draper answered 17 questions with a 71% accuracy. This is an increase of 21 percentage points from E2 with the teams answering 13 more questions in E4 than in E2. Draper had a higher "Yes" accuracy that "No" accuracy with a 32-percentage point difference in the two categories. Draper identified an unintended vulnerability in Airplan that impacted 5 challenge questions. In Draper's responses, based on our understanding of their research approach, it was unclear how the results from the AProVE tool on individual methods resulted in the responses without significant manual effort and unsound heuristics.

Table A-65: Engagement 4 Draper Labs Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	0	0	NA	0	NA
SC Time	2	1	50	1	50
SC Space/ Time	1	1	100	1	100
AC in Space	4	3	75	3	75
AC in Time	10	8	80	7	70
Total	**17**	**13**	**76**	**12**	**71**

Table A-66: Engagement 4 Draper Labs Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	13	10	**77**	9	**69**
Not Vulnerable	4	3	**75**	3	**75**
Yes Answer	11	10	**91**	9	**82**
No Answer	6	3	**50**	3	**50**

A.4.2.2 Draper Labs Specific Responses

A.4.2.2.1 Airplan 1

A.4.2.2.1.1 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Response: ""It is possible to upload a route map with a negative cost for a flight. This negative cost causes the method edu.cyberapex.chart.OptimalPath.calculateOptimalPath to loop indefinitely, which (of course) exceeds the timing requirements. The sequence of calls to reach the vulnerable method is:

flightplanner.guide.OptimalPathGuide.handlepost -> pullContents -> flightplanner.ChartAgent.getOptimalPath -> flightplanner.chart.OptimalPath.optimalPath -> calculateOptimalPath

Crucially, the *if* statement in the while loop that updates the path cost does not check positivity, nor does that check happen in the PartFileLoader.invoke() method that populates the cost field."

Evaluation: The shortest path implementation (A* search with an inconsistent heuristic) is vulnerable. Draper identified an unintended vulnerability in the application. Based on our understanding of Draper's research approach it is difficult to understand how the tool output of upper and lower bounds for individual methods led to the identification of this vulnerability without significant manual effort.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.1.2 Question 051 (AC Space, Unintended Vulnerable, Answered Yes)

Response: "The AC vulnerability in time translates to a vulnerability in space, as the 'calculateOptimalPath' method performs allocations within the infinite loop. In particular the

allocation of a PriorityNode is done on the heap. This allows us to exceed the memory budget within the allowed input."

Evaluation: Draper identified an unintended vulnerability in the application. Based on our understanding of Draper's research approach it is difficult to understand how the tool output of upper and lower bounds for individual methods led to the identification of this vulnerability without significant manual effort.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.2 Airplan 3

A.4.2.2.2.1 Question 024 (AC Time, Intended Vulnerable, Answered Yes)

Response: Draper reported the same vulnerability as Airplan 1 question 020.

Evaluation: The challenge program contains a vulnerable Ford Fulkerson implementation. Draper identified an unintended vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.3 Airplan 4

A.4.2.2.3.1 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

Response: Draper reported the same vulnerability as Airplan 1 question 020.

Evaluation: The challenge program contains a vulnerable Dijkstra's implementation. Draper identified an unintended vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.4 Airplan 5

A.4.2.2.4.1 Question 052 (AC Space, Intended Vulnerable, Answered Yes)

Response: Draper reported the same vulnerability as Airplan 1 question 020.

Evaluation: The challenge program is vulnerable to an XML bomb style attack. Draper identified an unintended vulnerability in the application.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.5 BidPal 1

A.4.2.2.5.1 Question 004 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "We used the java microbenchmarking harness to confirm that the runtime of MgProductGenerator.exponentiate significantly increases with the number of 1s in the exponent (see test/howto.txt to reproduce it and test/result.txt for the results). Since this method is used for encryption, this enables a side-channel attack. A trace that can be exploited by an attacker is:

BarterHandler.handle

-> BarterDriver.processOffer

-> BarterDriver.sendSahreData

-> new ExchangeData

-> ExchangeData.make

-> ExchangeData.generateHandler

-> EncryptionPrivateKey.decrypt

-> MgProductGenerator.exponentiate

Note that such an attack might be complicated by the hard to predict runtime of ExchangeData.make, which loops as long as randomly generated prime numbers are not good (which is determined by ExchangeData.verifySpacing). However, further tests showed that the critical loop in ExchangeData.make is usually evaluated once and only once for randomly generated data (see GoodPrime.java). Note, finally, that the UnbalancedBranches heuristic algorithm flags the fast exponentiation method. Hence, a side channel attack is still feasible."

Evaluation: The network communications framework coupled with a reduced budget is believed to effectively prevent a vulnerability. Given that described vulnerability leaks the number of 1s in the exponent it is unclear how the hamming weight of the exponent sufficiently reduces the entropy of the secret and the use of the method for encryption does not necessarily indicate the presence of a potential vulnerability. Based on our understanding of Draper's research approach it is difficult to understand how the tool output led to the identification of this potential vulnerability without significant manual effort and the application of a heuristic based on the method name.

Post-Engagement Analysis Ruling: Incorrect

A.4.2.2.5.2 Question 033 (AC Time, Unintended Vulnerable, Answered Yes)

Response: "AProVE can infer reasonably small bounds for most methods (see aprove_results.txt). Exceptions are

(1) ExchangeData.verifySpacing where AProVE infers a quadratic upper bound

(2) many constructors that read from InputStreams

(3) ItemList.toString and LoggerFactory.fixSubstitutedLoggers, where AProVE and all decompilers that we use failed

(4) BarterProtoSerializer.deserialize, where AProVE and all decompilers that we use failed

(5) ExchangeData.make, which has a very suspicious termination argument.

regarding (1): the critical argument is an array whose length is hard-coded to 500 for the suspicious traces that we found

regarding (2): these constructors are obviously linear

regarding (3): we assume that these methods are safe, but since we cannot decompile them, this is essentially an educated guess

regarding (4): if we missed a vulnerability, then it's most likely here

regarding (5): this can be exploited

To exploit (5), we created a modified version of bidpal_1.jar (see bidpal_1_bad.jar) where the instance of BidDivulgeData in makeBid is created as follows:

new BidDivulgeData(barterId, bid, BigInteger.ZERO, random.nextDouble(), random.nextDouble());

Here, it is important that the third argument is 0. The reason is that BidDivulgeData.fetchCommitmentData now creates an instance of BidCommitmentData where *shared* has the value 0 if the bid is 0. To achieve that, we decompiled BarterDriver, modified it accordingly, compiled it, and replaced BarterDriver.class in bidpal_1.jar. […]"

Evaluation: The challenge does not contain an intended vulnerability. Draper identified the same unintended vulnerability as Invincea. Based on our understanding of Draper's research approach it is difficult to understand how the AProVE output for ExchageData.make is flagged as potentially vulnerable and how this method is distinguished from other methods based on the AProVE output without significant manual effort.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.6 BidPal 2

A.4.2.2.6.1 Question 044 (AC Time, Intended Vulnerable, Answered Yes)

Response: Draper reported the same vulnerability as in BidPal 1 question 033

Evaluation: The challenge contains an intended vulnerability in the auction process. For a specific username and auctionID the application will continually generate failed checksums. Draper identified the same unintended vulnerability as Invincea.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.7 Collab

A.4.2.2.7.1 Question 035 (AC Time, Intended Not Vulnerable, Answered No)

Response: "The most suspicious functionality offered by collab w.r.t. DoS attacks is searching for events within a given range. However, the server reliably ensures that a single user must not add more than a fixed number (2500) of events. This is also ensured if the user logs in multiple times (i.e., renews the session) to add more events. Hence, we believe that the application is not vulnerable to DoS attacks. To justify our answer, we used our toolchain to analyze the complexity of collab's methods separately. Indeed, our toolchain is able to prove reasonable polynomial bounds for all but three methods. The methods where our toolchain fails are:

* client.CommandLineApp.main -- This method intentionally does not terminate.

* collab.CollabServer.channelRead0 -- Manual inspection shows that this method does not do any expensive computations and hence we consider it to be safe. Our toolchain fails since the big switch-statement massively blows up the state space for our symbolic execution.

* collab.dstructs.BTree.delete -- This method currently exceeds the capability of our tools. We inspected it manually and while it is complex, we do not believe it is vulnerable w.r.t. DoS attacks.

Apart from these three methods, the cubic upper bound that our tools inferred for collab.SchedulingSandbox.addhelper seemed suspicious. However, manual inspection shows that the complexity of this method depends on the size of eventidlist, which is small (initally at most 9). Hence, we consider this method to be safe as well."

Evaluation: Draper correctly determined that there was not an intended vulnerability. Based on this response it appears that there are many un-sound manual steps from AProVE's output to determining that an application is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.8 Malware Analyzer

A.4.2.2.8.1 Question 023 (AC Space, Intended Vulnerable, Answered No)

Response: "Here only one method writes to file: 'Server.serve()' (via the 'Logger.write_log_entry' method). The analysis is similar to the time analysis, where the output is largest for the 'get_cfg' requests. The log file is overwritten at each query. We do not believe that there is a way to get output incomensurable with the size of applications added with 'add_dasm'. In particular, we do not believe that it is possible to get 80 KB of output starting with an empty database and an input of 3000 bytes."

Evaluation: The challenge contains a vulnerability in that an integer overflow can be used to trigger a divide by zero exception. The resulting stack trace is written to a log file. On Draper's statement concerning "starting with an empty database." The challenge applications are often provided with an existing database. Challenge questions are based on the provided challenge applications in their un-altered form running on the reference platform. As a result, the state of the delivered challenge program is the state at which an attacker begins to interact with the application. In the case of Malware Analyzer, the challenge was provided with a given database of files. This is the state at which the attacker begins to interact with the application. From this response, it is unclear how Draper's tools aided in ruling out the challenge application as vulnerable.

Post-Engagement Analysis Ruling: Incorrect

A.4.2.2.8.2 Question 058 (AC Time, Unintended Vulnerable, Answered No)

Response: "Our complexity analysis identifies several methods with potential high resource usage:

com.ainfosec.MalwareAnalyzer.Server.createChecksum([B)[B -> Unknown Complexity

com.ainfosec.MalwareAnalyzer.Server.serve(Lfi/iki/elonen/NanoHTTPD$IHTTPSession;)Lfi/iki/elonen/NanoHTTPD$Response; -> Unknown Complexity

com.ainfosec.MalwareAnalyzer.DasmDatabase.compute_all_similarities(Ljava/lang/String;)Ljava/util/NavigableSet; -> Quadratic Complexity

com.ainfosec.MalwareAnalyzer.DasmHelpers.make_function_cfg(Ljava/util/TreeMap;Ljava/lang/Integer;)Lcom/ainfosec/MalwareAnalyzer/CFG; -> Unknown Complexity

The other functions are either recursive or determined to be constant or linear. The only interesting recursive function we identified is the 'Analysis.compute_cosine_similarity' Of these, the serve function is the most complex. One of the cases of the switch function prints a representation of the whole call graph, which may be quadratic in the number of functions. However, it is unlikely to be the source of a vulnerability: the worst case is a program in which every function calls every other function. But this requires a quadratic size in the number of functions to write. We do not believe that there is a way to exceed the resource usage limit using fewer than 3000 bytes."

Evaluation: Invincea identified an unintended vulnerability. It is unclear from this response how other recursive methods beyond the identified “Analysis.compute_consine_similarilty” were ruled out based on AProVE’s output without significant manual effort.

Post-Engagement Analysis Ruling: Incorrect

A.4.2.2.9 PowerBroker 1

A.4.2.2.9.1 Question 019 (AC Space, Intended Vulnerable, Answered Yes)

Response: “The IDs for cities are shortened to 25 characters for logging in most, but not in all cases. This can be exploited by an attacker. To see this, run powerbroker_1/exploit_19/exploit.sh, which results in huge log files.”

Evaluation: The challenge contains a vulnerability in the connection establishment. If an attacker terminates a connection before it is complete, the user will continuously attempt to reconnect, logging each failed attempt. Draper identified the same unintended vulnerability as Northeastern. Based on our understanding of Draper’s research approach it is difficult to understand how the tool output of upper and lower bounds for individual methods led to the identification of this vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.10 RSA Commander

A.4.2.2.10.1 Question 039 (AC Time, Intended Vulnerable, Answered Yes)

Response: “There is a *while* loop in 'stac.communications.Communications.doHandshake which may run *forever* (in particular more than 2 hours) if the result of 'HandshakeHandler.handle' always returns HANDLER_STATE.WAITING. For this to happen, the stac.communications.Handler.handle methods needs to always return HANDLER_STATE.WAITING. This happens if the branch depending on isPacketStillOK(PacketBuffer) is hit and returns true, isPacketStillOK always returns true for instances of RequestHandler […] RequestHandler.isPacketFormed needs to return *false* on a packet built from the remote input […] The complexity analysis tool identifies 'stac.communications.Communications.sendMessage' as a method with potential infinite running time. The taint analysis shows that the address of the attacker can potentially reach the 'stac.communications.Handler.handle' InputStream argument, which on inspection is controlled in a strong way by the remote client. […]”

Evaluation: This was an unintended vulnerability reported in E3. The *additional information* section of E4 variant of the challenge question prevents this vulnerability. Based on our understanding of Draper’s research approach it is difficult to understand how the tool output of upper and lower bounds for individual methods led to the identification of this potential vulnerability without significant manual effort.

Post-Engagement Analysis Ruling: Incorrect

A.4.2.2.10.2 Question 053 (SC Time, Intended Not Vulnerable, Answered No)

Response: “The message to send is (also) processed and encrypted in 'Communications.sendMessage' simply using the call-graph and reading the decompiled code, we can find that the succession of methods used to process each byte of the message are, in order

Communications.sendMessage → RequestPacketParser.serialize → RequestPacketParser.appendRijndaelEncryptedSessionContent → DES.encrypt_ctr → DES.encrypt → DES.encryptBlock → DES.getBytesFromLong

For each of these methods, we can use the run-time upper *and lower* bounds to see if there seems to be a discrepancy between the longest and shortest run-time for these methods. Note that if a discrepancy exists, we have no guarantee of a vulnerability or vice-versa. However, most methods are seen to have identical bounds, meaning that the run-time is not sensitive to the input, except for the 'getBytesFromLong' method. However, we do not believe that the time behavior of this method is sufficient to give information about the bytes from the original method. We conclude that there is no side channel vulnerability which reveals the message, but without formal guarantee."

Evaluation: Draper correctly determined that there was not an intended vulnerability. Draper's response indicates a significant manual effort prior to identifying methods to run on. Additionally, the team reported only the "getBytesFromLong" method as sensitive to the input, but the RSA encryption implementation leaks the nonce and counter values that can be used to decrypt the DES-encrypted message. The leaks from the RSA implementation is insufficient as it does not satisfy the operational budget for the challenge question.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.11 SmartMail

A.4.2.2.11.1 Question 047 (SC Time/Space, Intended Vulnerable, Answered Yes)

Response: "There is an SC space vulnerability in SmartMail that can reveal the email address of a SmartMail mailing list administrator. When sending an email to an administrator, the SmartMail server appends an encrypted string to the admin log file logs/addlog1.log. Appending to the log file will increase the size of the file, which the attacker can observe. The attacker can also observe non-administrator emails in logs/addlog2.log. Hence, sending an email to an address where file logs/addlog1.log increases in size will confirm to the attacker that the destination email address is indeed an administrator. The attacker does not know a-priori the list of all emails in the SmartMail application. However, an attacker can send an email to an administrator by emailing a mailing list that the administrator is subscribed to. The attacker can determine the indices of administrator emails into the mailing list. Emails are sent to members of a list in alphabetical order of the email addresses. (which are prefixes of the keys to the map MailingList.members). The attacker can observe the order of when the admin log file logs/addlog1.log and the public log file logs/addlog2.log are appended to. […]"

Evaluation: Draper identified the intended vulnerability. It is unclear how Draper's tools aided in the identification of this vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.2.2.11.2 Question 059 (AC Time, Unintended Vulnerable, Answered No)

Response: "The application was determined to not be vulnerable to an AC vulnerability in time. Size limits are enforced for SmartMail queries, including number of addresses, size of addresses and e-mail body size. The complexity analysis verified the complexity of most methods, the remainder were checked by hand. Only smartmail.process.controller.module.StateHolder.callDumper(Ljava/lang/String;)V was found to

be quadratic, and hand examination of the call stack to this method showed that this was insufficient to cause an AC vulnerability in time."

Evaluation: Northeastern and Utah identified an unintended vulnerability. Draper's response indicates significant manual effort to rule out vulnerabilities.

Post-Engagement Analysis Ruling: Incorrect

A.4.2.2.12 Tour Planner

A.4.2.2.12.1 Question 056 (AC Time, Intended Not Vulnerable, Answered No)

Response: "There is NO AC time vulnerability for application tour_planner. The set of places and their locations are stored in file tour_planner/challenge_program/data/matrix.csv. The number of places is 25, which is a small size for polynomial time running algorithms. The algorithm performed by the server for computing tours is implemented in method com.graphhopper.tour.TourCalculator.calcTour. Determining which method was being executed for calculating tours was performed by using our taint analysis tool (which generates a call graph). We verified that all tainted methods (methods processing user input) had polynomial running times (even though the *exact* solution to the TSP is NP-hard). The algorithm for calculating tours implemented in tour_planner is a heuristic algorithm rather than optimal algorithm. The heuristic algorithm is a well-known algorithm that computes tours by computing the minimum spanning tree. The minimum spanning tree algorithm running time is polynomial in the number of places in the tour. After the minimum spanning tree is computed, a depth-first search traversal over the minimum spanning tree is performed to compute a walk/tour over nodes/places in the tree. Depth-first search traversal runs also polynomial time in the number of places in the tour. We used our complexity analysis tool AproveInvoker to investigate runtimes of methods that are reachable com.graphhopper.http.TourServlet.doGet in the function call graph. AproveInvoker was able to infer linear runtime upper bounds for almost all methods in the subgraph of methods reachable from method doGet. The methods which could not be automatically analyzed were calcMinSpanningTree and depthFirstWalk. However, since AproveInvoker flagged polynomial bounds for most other methods, we were able to substantially narrow the scope of manual investigation. We manually determined the polynomial runtime bounds of these key methods. We can conclude all reachable methods from doGet have polynomial runtime: Linear for all methods except calcMinSpanningTree. As a side note, the HashSet used for storing the points in the query could hypothetically take quadratic time (in the number of points) to be built. We verified that this was not the case, as no pairs of points have equal hashcodes. This, combined with the fact that there are no exessively expensive constant-time operations allows us to conclude that there are no AC time vulnerabilities."

Evaluation: Draper correctly determined that there was not an intended vulnerability. Based on our understanding of Draper's research approach it is difficult to understand how the tool output of upper and lower bounds for individual methods led to ruling out this vulnerability without significant manual effort.

Post-Engagement Analysis Ruling: Correct

A.4.3 GrammaTech

A.4.3.1 GrammaTech Overview

GrammaTech answered 22 questions with a 91% accuracy, a 5 percentage point increase from E2 and a 35 percentage point increase from E3. GrammaTech answered 8 more questions in E4 than in E2 answering more questions in every category except SC Time. The team's accuracy decreased by 11 percentage points in SC Space from E2; however, this category accounted for the team's increase in responses in E4 compared to E2. GrammaTech had the highest R&D Team accuracy and TPR against the segmentation of answered questions, and the second highest TNR against the segmentation of answered questions.

Table A-67: Engagement 4 GrammaTech Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	8	89	8	89
SC Time	1	1	100	1	100
SC Space/ Time	0	0	NA	0	NA
AC in Space	6	5	83	5	83
AC in Time	6	6	100	6	100
Total	**22**	**20**	**91**	**20**	**91**

Table A-68: Engagement 4 GrammaTech Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	15	14	**93**	14	**93**
Not Vulnerable	7	6	**86**	6	**86**
Yes Answer	15	14	**93**	14	**93**
No Answer	7	6	**86**	6	**86**

A.4.3.2 GrammaTech Specific Responses

A.4.3.2.1 Airplan 1

A.4.3.2.1.1 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Response: "The methods `edu.cyberapex.chart.Limit.search()` and `edu.cyberapex.order.shifter.changingArrange()` are involved in an ACV-T. […] This method performs a breadth-first exploration of a graph. Each loop iteration calls the method `pullEdges()`, which sorts a list of edges using a modified merge sort algorithm. The core of that sorting algorithm is `edu.cyberapex.order.shifter.changingArrange()` […] A worst-case scenario that forces a large input for the search algorithm is a graph containing nodes with a high out-degree."

Evaluation: The challenge contains a vulnerability in the A* implementation. The algorithm uses an inconsistent heuristic resulting in unnecessary traversals. GrammaTech identified an unintended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.2 Airplan 2

A.4.3.2.2.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)
Response: GrammaTech provided the same exploit as for Airplan 1 question 020.

Evaluation: The challenge contains an intended vulnerability in the sort implementation. If the number of items to be sorted is divisible by 6, there are unnecessary operations added to the merge queue. GrammaTech identified an unintended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.2.2 Question 038 (SC Space, Intended Vulnerable, Answered Yes)
Response: “When the victim uploads a new route map, the server redirects to the Map Properties page and then to the Flight Matrix Page. The size in bytes of this Flight Matrix page is an observable, and is correlated with the secret, i.e., the number of airports in the map. […]”

Evaluation: GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.3 Airplan 3

A.4.3.2.3.1 Question 002 (SC Space, Intended Vulnerable, Answered Yes)
Response: “When the victim submits a new route map, the server redirects to the "Map Properties" page that contains information about the submitted map, including whether it is strongly connected. The method of interest is `net.cybertip.routing.manager.MapPropertiesCoach.getContentsForPost()`. The observable is the size of the HTML page returned by the server. The secret is whether the map is strongly connected or not. The observable relates to the secret because the size of the returned HTML page is going to be different depending on whether the map is strongly connected or not. […]”

Evaluation: GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.3.2 Question 016 (SC Space, Intended Not Vulnerable, Answered No)
Response: “When the victim uploads a new route map, the server redirects to the Map Properties page and then to the Flight Matrix Page. The size in bytes of this Flight Matrix page is an observable. The way the page is constructed depends on the secret, i.e., the number of airports in the map. There could be a side channel if an attacker can learn the number of airports in a victim's map just by looking at the size of the Flight Matrix HTML page. The relevant code is in `net.cybertip.routing.manager.FlightMatrixCoach.handleObtain()`. This method calls `routeMapAsTable()` to generate the HTML page returned to the victim. `routeMapAsTable()` builds the page from constant-size Strings and from the String returned by the `generateRows()` method. This last method builds a square table of width and length equal to `airports.size()+1` (where `airports.size()` is the secret). However, the cells of this table have randomized length:

they are created using the `format` method, and its `adjust` parameter is set to `true`. Consequently, each cell will have a size `len` computed using a random generator […]"

Evaluation: GrammaTech correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.4 Airplan 4

A.4.3.2.4.1 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

Response: GrammaTech provided the same export as Airplan 1 question 020 and Airplan 2 question 037.

Evaluation: The challenge contains an intended vulnerability in the Dijkstra's algorithm implementation used to calculate the shortest path. GrammaTech identified an unintended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.4.2 Question 055 (SC Space, Intended Not Vulnerable, Answered No)

Response: GrammaTech repeated the analysis for Airplan 3 question 002 and concluded that the application is not vulnerable.

Evaluation: GrammaTech correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.5 Airplan 5

A.4.3.2.5.1 Question 005 (SC Space, Intended Not Vulnerable, Answered No)

Response: GrammaTech repeated the analysis for Airplan 3 question 016 and Airplan 2 question 038 and concluded that the application is not vulnerable.

Evaluation: GrammaTech correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.6 Collab

A.4.3.2.6.1 Question 028 (SC Time, Intended Not Vulnerable, Answered No)

Response: "[…] The audit id filtering logic in `getRange()` does not depend on the specific audit id. Therefore, any attack based on the `getRange()` method would have to rely on some sort of binary search varying the search range. However, `getRange()` actually ignores the passed-in range bounds `min` and `max` and searches the tree for the full possible range of event ids, i.e., 1 to 2147483646. Therefore, a side channel attack of the kind we require cannot be achieved with `getRange()`. […] As for the binary search attack based on node splits and `add()` invocations, it is not possible to achieve it within a budget of 200 operations. The range of possible values for an audit id is 2147483647 - 100000. Thus, narrowing down the audit ids to a single value would require at best $\log_2(2147483647 - 100000) \cong 31$ steps. The attack would require inserting at least 8 new ids at every step and $8 * 31 > 200$ operations. Furthermore, we measured empirically the timing differences in the split vs no split case, but the difference was too small to observe. This strengthens our conclusion that there is no way to obtain a timing side channel using `add()` operations on the sandbox. […]"

Evaluation: GrammaTech correctly determined that there was not an intended vulnerability. The time distribution for split and non-split operations are indeed sufficiently overlapped; however, if this were not the case, for the provided public data (Picard's events) there would be a sufficient number of operations to extract the secret.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.7 Linear Algebra Platform

A.4.3.2.7.1 Question 034 (AC Space, Intended Vulnerable, Answered Yes)

Response: "For the shortest path computation, `com.example.linalg.external.LinearAlgebraService` invokes ` com.example.linalg.external.operations.ShortestPathOperation` which invokes `com.example.linalg.util.MatrixSerializer.readMatrixFromCSV` twice. For the second call, bounds checking on the matrix is turned off and the program allocates an array of arbitrary size based on the passed-in `rows` and `cols` values in the JSON file. Note that the JSON file does not actually have to contain enough CSV data to match the numbers of rows and columns in the `rows` and `cols` field, as that check happens after the array allocation […]"

Evaluation: GrammaTech identified the intended vulnerability. Note: in the E3 variant of this challenge, this was an unintended vulnerability discovered by GrammaTech and 1 other team. Given the realism of the vulnerability, it was included in E4 as an intended vulnerability rather than removing it.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.8 Malware Analyzer

A.4.3.2.8.1 Question 023 (AC Space, Intended Vulnerable, Answered No)

Response: "We identified two potentially vulnerable code structures in malware_analyzer. The first structure involves the call to the method `printStackTrace` from `Server.java`. The attacker can cause this statement to print a long traceback; the traceback is then copied into the `Server.response` variable, which is eventually written to disk. To produce a long traceback, the attacker can cause an error in one of the deeply recursive methods in Analysis.java. The second potentially vulnerable code structure is the part of `DasmHelpers.java` that takes a piece of disassembled code as input and produces a control-flow graph as output, along with the code in `Server.java` that prints a control-flow graph and ultimately writes it to a file via the `Server.response` variable. […] For the first potentially-vulnerable code structure, we manually constructed inputs in an attempt to cause an exception in `Analysis.java` that would cause a long traceback to be printed. The longest traceback that we were able to produce was approximately 20000 bytes. We produced that traceback by causing a divide-by-zero exception in the 255th recursive call of one of the methods named `compute_cosine_similarity_helper`. We also calculated that if we had been able to cause an exception in a different method that has even deeper recursion, we would still only have been able to produce a traceback of approximately 60000 bytes. For the second potentially-vulnerable code structure, we manually analyzed the decompiled code of `DasmHelpers.java` to determine whether we could cause large data structures to be created. The `make_function_cfg` method has two potential vulnerabilities. First, it is possible for an attacker to cause single instructions of assembly code to be interpreted as belonging to multiple functions. As a result, an attacker can cause `make_function_cfg` to build a data structure whose size is a quadratic function of its input. However, there is no single query

that an attacker can issue to cause this large data structure to be printed to a single file, so this attack does not cause a large file to be written to disk.

A second potential vulnerability in `make_function_cfg` relates to the ability of the attacker to cause single instructions of assembly code to be identified as part of multiple basic blocks. If the attacker could arrange for this, then the attacker could send a `get_cfg` query to the malware_analyzer server, and the server would print the entire control-flow graph, including the repeated instructions, to a file. However, this potential vulnerability is not suitable for an attack because the code of `make_function_cfg` uses a map (called `assigned_instructions`) that specifically prevents a single instruction from being assigned to more than one basic block. […]"

Evaluation: This challenge contains an intended vulnerability. An integer overflow can be used to trigger a divide-by-zero exception. The resulting stack trace is then logged to a file on disk. GrammaTech's response indicates they flagged the calls to printStackTrace as a potential vulnerability. They identified a divide-by-zero exception in compute_cosine_similary_helper, but their analysis indicates that they did not flag the potential for integer overflow in the calculation of average byte value.

Post-Engagement Analysis Ruling: Incorrect

A.4.3.2.9 SmartMail

A.4.3.2.9.1 Question 043 (AC Space, Intended Not Vulnerable, Answered No)

Response: "The attacker is only allowed to perform three operations: receive an email as a given user, get the list of public subscribers for a mailing list, and send an email. Receiving an email as a given user cannot increase the memory usage of the program, in fact it reduces it once garbage collection occurs, since the email is removed from a queue in `smartmail.datamodel.MailBox.java` […] Getting the mailing list of the subscribers for a list has a memory usage dependent on the number of subscribers to the list, since the addresses of all subscribers are collated into a `StringBuilder` in `EmailSenderReceiver.getAddress()` […] This could potentially cause large memory consumption if a list has a huge number of subscribers. However, based on clarifications provided to us, we understand any ACV must manifest with the provided canned data. There is no way to exceed 1024MB of memory usage with the provided mailing list, where the maximum number of subscribers to any list is 6. This leaves the option for the attacker to cause excessive memory consumption via sending one or more emails. However, the method `EmailSenderReceiver.doEmail()` has strict checks on input size […] A single email has an effective limit of 10 "to" addresses. Furthermore, with the canned data provided there are only six users in the system, and the map-reduce logic that sends emails removes duplicate emails (that might occur e.g. if the attacker placed the same email address in the "to" field twice). Thus, every email-send operation delivers emails to at most six mailboxes. The total attacker input budget is 10000 bytes. Some of that budget must go towards specifying email addresses rather than payload of messages. However, even if all 10000 bytes could be used towards an email body, by the above argument the attacker could only cause 6 x 10000 bytes = 60KB of email subject/content data to be stored on the server. Even allowing for some extra memory consumption due to metadata, 60KB is well under the desired attack threshold of 1024MB."

Evaluation: GrammaTech correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.10 Tour Planner

A.4.3.2.10.1 Question 011 (SC Space, Intended Not Vulnerable, Answered No)

Response: "We arrived at this conclusion through manual inspection of the decompiled code, with a focus on all methods that our heuristics flagged as being of potential high complexity. […] The observables are the packets sent and received by some user and their size. The relevant code that might contain an SCL-S is `doGet` in `com.graphhopper.http.TourServlet.java`. The input `req` of type `HttpServletRequest` has a size that can be considered as an observable (the attacker can see the size of the packets sent by the user to the server). This `req` object contains the secret, which is the list of points that the user wants to visit: in the code, it is the list `List<GHPoint> points`. On the other hand, the second argument `HttpServletResponse res` can also be considered as an observable: one can look at the size of the packets sent by the server to the user, which gives information about what is written to `res`. An analysis of the code shows that some progress reports are written to `res` through the `progressReporter` object, as well as the JSON serialization of the tour computed by the server. The progress reports can be easily shown to have a constant size, and thus are not relevant to an SCL-S. However, the JSON serialization of the tour will have a size that varies depending on the length of the city names in the secret list `points`. The serialization of a point into JSON is for instance `{"name":"Peabody","lon":-70.9286609,"lat":42.5278731}`. In the provided list of cities, the fields "lon" (resp. "lat") are double values with always the same string size 11 (resp. 10). This means that a point is always serialized into a string of length `46 + length of the city name`. From there, the observable reveals the sum of the length of the cities in the secret (the city that starts and ends the tour is counted twice). One can send a tour request to the server and watch the packets […] The only two packets with a size that varies depending on the secret are packets number 13 (containing the GET arguments, i.e. the secret list of points) and 24 (containing the JSON serialization of the tour, which also contains the name of the cities in the secret list). The other packets are either ACK packets or progress reports of constant size. […] It is possible to reveal some information about the secret by looking at the size of the two packets mentioned above. More precisely, these two packets can reveal information as follows:

1. The first packet reveals the sum of all string lengths of the 5 cities in the secret list.

2. The second packet reveals the sum of all string lengths of the 5 cities in the secret list with the origin/end counted twice.

3. Indirectly from 1. and 2., the two packets reveal the string length of the first/last city in the tour.

NOTE : 1. and 3. assume that the only GET arguments sent by the user are the list of points. If this is not the case (e.g. the user specifies the same point multiple times, or adds some other useless arguments), the only information we the attacker has is 2. In the following argument, we assume 1. and 3. are true; this is the easiest scenario for the attacker. Even in that case we can prove that there is no SCL. The attacker can also obtain the list of all cities in the server's matrix with a `places` query. Since the budget is 3000 operations, and only one of these operations is used to obtain the size of the two interesting packets sent/received by the victim, the attacker has 2999 operations left to do oracle queries. The question is then : given the size of these two packets, how many sets of 5 cities will have exactly the same packet size? If there is less that 2999 possibilities for any possible packet size, then there is an SCL of sufficient strength. The number of possibilities obviously depends on the list of cities in the matrix. Any attack is

required to work on the provided example matrix. This matrix contains 1 city with name size = 4, 6 with name size = 6, 4 with name size = 7, 5 with name size = 8, 4 with name size = 9, 3 with name size = 10, and two cities with name size = 11.

We used a python script that uses z3 to provide the exhaustive list of oracle queries the attacker should perform given this matrix, and given the integer values of 1. and 2. described above […] Depending on the values described in 1. and 2. above, the attacker's budget of 3000 operations might or might not be sufficient to succeed with a probability of 99%. For instance, if the value in 1. is equal to 40 and the value in 2. is equal to 48 (meaning that the origin/final city has 8 characters and the total number of characters in the cities' names is 48), there are 4196 distinct sets of 5 cities for which the attacker needs an oracle query, which significantly exceeds the budget of 3000. Note that many pairs of values from 1. and 2. would require less than 3000 queries and in that case the attacker would succeed. However, the overall probability of success will be less than 99% in general."

Evaluation: GrammaTech correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.11 Tweeter

A.4.3.2.11.1 Question 029 (AC Time, Intended Vulnerable, Answered Yes)

Response: "This code is the implementation of spell-check logic in `com.tweeter.service.Spelling`. Every word in a tweet is passed to method `correct()` […] This method calls `isPossible()` and `getCorrections()`. Both of these methods check that the word is within [Damerau-Levenshtein](https://en.wikipedia.org/wiki/Damerau%E2%80%93Levenshtein_distance) edit distance 2 of some word in the provided dictionary. The check is performed by generating an exhaustive set of all possible words within edit distance 1, then edit distance 2, and checking whether any of those words is in the dictionary. Each word is checked against the dictionary as it is generated, but if there are no dictionary matches, the full set of words at edit distance <=2 will be generated. For a very long word w, the set of all possible words which are 1 or 2 edit distance away from w will be large. Generating such a set and verifying whether any word in the set is in the dictionary could take time that exceeds the resource budget. The method `correct()` will reject any word whose length is greater than MAX_WORD_LENGTH + 2, where `MAX_WORD_LENGTH` is the length of the longest word in the provided dictionary. This word is "electroencephalographies", with length 24. However, we cannot achieve the worst-case runtime with any arbitrary string of length 26. This is because the method `isPossible()` performs filtering checks on all prefixes of the word. If it finds that any prefix is not within edit distance 2 of a prefix of a dictionary word, it will terminate and return false. Thus, the worst-case input must be a variant of a long dictionary word with only the final few characters misspelled. On such an input, the prefix filter check will pass for all short prefixes of the word. […] We discovered the vulnerability through manual inspection of the decompiled code. We also applied ICRA to a modified version of the code in an attempt to verify the attack independently; however, we were not able to obtain adequate results before the engagement deadline."

Evaluation: GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.12 WithMi 1

A.4.3.2.12.1 Question 017 (AC Time, Intended Vulnerable, Answered Yes)

Response: "We believe there is a vulnerability in the method `expand` of `edu.networkcusp.smashing.SpiffyPacking`. The loop below executes `length` times, where `length` is an input. It is possible for an attacker to craft an input that makes this loop iterate sufficiently many times to exceed the time budget. […] The method `expand` is supposed to decompress a compressed file that has been sent by another user using the command `sendfile`. The compression is based on the Huffman encoding. The first part of the compressed file is the encoding of a tree structure, followed by an integer `length` specifying the size of the compressed data, followed by the compressed content of the original file using this tree. If the `length` value that is read from the input stream is very large (e.g. if length = MAX_INT), the running time of the entire loop will be large. The inner loop `while (!x.isLeaf()) {...}` reads from the input (`in.readBoolean()`), which might suggest we require a large input budget to ensure a large number of iterations of the outer loop. However, it is also possible that `x.isLeaf()` is always true and that the inner loop body does not execute. This happens if the input Stream that is processed by `readFormation()` contains only one Boolean that equals True. Also, if the inner loop body does not execute, the integer variable `read` is never incremented, and the test `if (read > 50000000) {...}` always evaluates to false : no exception will be thrown. In this situation, the outer loop can execute MAX_INT times, without reading any input bytes. Within each iteration, it calls `IntegratedOut.write` at each iteration, which is expensive as it involves a triply-nested while loop. […]

An attacker can pretend to send a file to a victim, and craft a compressed file that will expose the vulnerability. A normal use of the withmi binary does not allow the crafting of such an attack, but the attacker can use a modified version of withmi. The compressed data that will cause the vulnerability is the following :

- 4 bytes that encode an integer. It will be read by `final int paddedMarker = in.readInt()`.

- The encoding of a Boolean 0. It will be read by `final Node root = this.readFormation(in,0)`. The `root` node will be a leaf.

- 4 bytes that encode the integer MAX_INT. It will be read by `final int length = in.readInt()`.

- No further data is required, as the loop won't read any other bytes from the InputStream.

This compressed file is much smaller than the input budget, and will cause the time budget to be exceeded on the victim's side.

Our heuristics identified `expand` as a potential high-complexity method. Manual inspection of the method led us to our conclusion."

Evaluation: GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.12.2 Question 032 (SC Space, Intended Vulnerable, Answered Yes)

Response: "When a user sends a chat message, the method that transmits the message to the other users in the chat is `deliverMessage` in the class `edu.networkcusp.chatbox.HangIn`. […] the secret is the size of the list `members`: its value gives the number of users involved in the chat the victim is currently in. The observable is the number and the size of packets sent by the

victim. These packets are sent by the `this.deliverMessageWorker` method call. […] If the chat has N users, the above code fragment will generate and send exactly N packets with the same size. This is due to the `for` loop that iterates over the members in the chat, and for each of them, builds and sends a packet of the same size. […] The attackers can just look at the packets sent by the victim and identify how many packets of the same size are sent in a short amount of time. Note that the time between the packets is randomized, but we believe it is still possible to guess the number of users, since the victim will still send the packets in sequence. […] The vulnerability was found by manual inspection of the decompiled code."

Evaluation: GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.13 WithMi 3

A.4.3.2.13.1 Question 030 (AC Space, Intended Vulnerable, Answered Yes)

Response: "The code containing a vulnerability is the method `reconstitute` in `net.robotictip.compression.SpiffyUnpacking.java`. […] This code is executed by a user when he/she receives a file that was sent using the `sendfile` command. […] The outer loop in this method iteratively reads a byte `b` and then an integer `num`. This integer is supposed to tell how many successive occurences of `b` should be written to the uncompressed file. The inner loop then writes these `num` occurrences of `b` into the output. However, the loop condition is `while (num != 0)`. This assumes that `num` is always positive, and that decrementing `num` inside the loop will eventually terminate. If the `num` value that is read from the input happens to be negative, the loop will never terminate and create an ACV in space (`b` will be written to `outStream` indefinitely) and time. […] A normal use of withmi that sends a file to another user will not cause the ACV, because the way the file is compressed does not permit `num` to be negative. However, an attacker could use a modified version of the withmi program to craft a compressed file such that `num` is negative, and send this file to a victim using the same protocol as a normal `sendfile` command. The "compressed" file can be as small as 5 bytes, the last 4 bytes encoding a negative integer, which is much less than the input budget. […] Our heuristics identified the `reconstitute` as containing both input and output logic, and therefore being potentially relevant to ACV-S vulnerablities. We arrived at our conclusion through manual inspection of the decompiled code."

Evaluation: GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.14 WithMi 4

A.4.3.2.14.1 Question 010 (SC Space, Intended Vulnerable, Answered Yes)

Response: "We focus on the `sendfile` command, not on `sendfilezlib`. Even though we do not give details about `sendfilezlib`, we believe the same SCL exists. The only difference is how the size of the packet relates to the file content. In the following, we describe the SCL with `sendfile`. The fact that 1) the size of the packets directly depends on the file content and 2) that the time between the packets is not randomized, give sufficient information to guess which file has been sent by the victim. One can look at the files within the directory `challenge_program/data/files`, and use the following script to see what the packets will look like if they are transmitted. […] Now, if we look at the wireshark capture of the packets sent by the

victim, we see the same sequence of packets in a very short time interval, with the constant value being 200. We ignore the last packet, but the other packets are sufficient to be confident enough. The vulnerability was discovered by manual inspection of the decompiled code."

Evaluation: GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.14.2 Question 036 (AC Space, Intended Vulnerable, Answered Yes)

Response: "We believe there is an ACV-S vulnerability in the method `com.digitalpoint.togethertalk.FileTransfer.receive()`. […] This invokes `WrapperManager.stretch()`. When the SNEAKER compression is used (e.g. when a file is sent by the withmi binary using a `sendfile` command), this call to `stretch()` executes `com.digitalpoint.smashing.SneakerWrapper.stretch()` […] These two methods together can be exploited to demonstrate an ACV-S. […] The `receive` method processes the packets a client receives when another user sends a file using the `sendfile` or `sendfilezlib` withmi command. For a given file transfer, the method `receive` is invoked multiple times - once each time a packet containing part of the compressed file is received. We assume that the compression algorithm in use is SNEAKER, so the `stretch` method that is invoked is the one shown above. When the user receives a packet that contains part of a compressed file, the `receive` method writes the content of this packet to `compressedOutputStream` […] This `compressedOutputStream` writes to a temporary file on disk `<filename>.compressed`. Then, `WrapperManager.stretch()` reads `<filename>.compressed` and decompresses it to `<filename>` using the object `uncompressedOutput` […] This appends new content to the file `<filename>`.

If we look at the code of `stretch` above, we see that one call to `stretch` can write up to 100000000 bytes to the decompressed file. The shortest InputStream that triggers this worst-case behavior contains just five bytes. The first byte can be any arbitrary value and the 4 next bytes encoding the integer 99999999. This InputStream is obtained as `fileMsg.getContent().substring(0, bytesToUse)` in the receive method. `fileMsg` is defined as `final Chat.FileMsg fileMsg = msg.getFileMsg();`. Therefore, it can be controlled by the attacker. One call to the `stretch` method cannot exceed the budget of 120MB because of the guard `if (totalSize > 100000000) {throw new IOException("Decompressed data exceeded maximum allowed size ");}`. However, two calls to `stretch` with the same OutputStream can exceed the budget. An attacker can craft a sequence of packets and send them to a victim, pretending it is sending a file using the `sendfile` command of withmi. A normal version of withmi would not allow crafting the appropriate packets, but we assume that the attacker can use a modified version of withmi. The attacker can send a first packet to a user, with the correct metadata, such that :

- `msg.getType() == FILE`,

- `fileMsg.getZlibCompression() == WrapperManager.Algorithm.SNEAKER`,

- `fileMsg.getFileName() == attack.txt`

- `fileMsg.getCurrentOffset() == 0`

- `fileMsg.getTotalSize() == ` a large value, say 10000. This is to make sure the integer `bytesToUse` is not 0.

- Moreover, the attacker should create a packet content such that `fileMsg.getContent()` contains the 5 bytes mentioned above: the first byte can be any positive integer and the next four bytes should encode the integer 99999999. When the victim receives this packet, it will execute `receive` and `stretch`. This will create a new file `attack.txt` with a size of ~100MB. This does not exceed the budget yet. The attacker can then send a second packet to the same user, with the same metadata and content, except `fileMsg.getCurrentOffset() != 0`, which will prevent the deletion of the already existing file `attack.txt` After processing that second packet, the victim's file `attack.txt` should be ~200MB in size, which exceeds the budget. Sending the two malicious packets only requires a very small input budget, and is below the limit of 10KB."

Evaluation: The challenge contains an AC vulnerability due to an integer overflow vulnerability during decompression. GrammaTech identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.15 WithMi 5

A.4.3.2.15.1 Question 014 (AC Space, Unintended Vulnerable, Answered Yes)

Response: GrammaTech provided the same response as WithMi 4 question 036.

Evaluation: The challenge does not contain an intended vulnerability. GrammaTech identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.3.2.15.2 Question 046 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "When a user sends a chat message, the method that transmits the message to the other users in the chat is `transmitMessage` in the class `com.techtip.chatbox.DropBy`. The interesting code fragment is the following:

```
final List<WithMiUser> customers = forum.fetchCustomers();
for (int p = 0; p < customers.size(); ++p) {
  final WithMiUser customer = customers.get(p);
  if (customer.hasConnection()) {
   this.transmitMessage(data, customer.getConnection());
  }
  else {
   this.printCustomerMsg("Couldn't find connection for " +
customer.pullName());
  }
}
```

Here, the secret is the size of the list `customers`: its value gives the number of users involved in the chat the victim is currently in. The observable is the number and the size of packets sent by the victim. These packets are sent by the `this.transmitMessage` method call. […] If the chat has N users, the above method will send exactly N packets with the same size, with a constant time interval between these N packets. This is due to the `for` loop that iterates over the members in

the chat, and for each of them, builds and sends a packet of the same size. […] The attacker can just look at the packets sent by the victim and identify how many packets of the same size are sent in a short amount of time. […] The vulnerability was found by manual inspection of the decompiled code."

Evaluation: This challenge does not contain an intended vulnerability. GrammaTech reported the same potential unintended vulnerability as Vanderbilt and Iowa State, though adding that the time frame for these packets can be used to identify the packets relevant to the number of users in the chat. The challenge question does not place further restrictions on the chats of the victim user. As a result, the user could be engaged in chats with two other users, and the number of chat messages would not be sufficient to disambiguate whether the user is in one chat with the two other users or in separate chats with each user.

Post-Engagement Analysis Ruling: Incorrect

A.4.3.2.16 WithMi 6

A.4.3.2.16.1 Question 054 (AC Time, Unintended Vulnerable, Answered Yes)

Response: GrammaTech provided the same response at WithMi 1 question 017 adding that: "The vulnerable method is `net.computerpoint.wrapper.SpiffyWrapper.unzip()`".

Evaluation: GrammaTech identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4 Iowa State University

A.4.4.1 Iowa State Overview

Iowa State responded to 20 questions with a 75% accuracy; a 6 percentage point increase from E2 where they answered 13 questions and a 42 percentage point increase from E3 where they answered 12 questions. The team had the greatest increase in "No" responses from E2 to E4 with the team providing 8 "No" responses with 75% accuracy for E4 versus 0 "No" responses in E2.

Table A-69: Engagement 4 ISU Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	7	88	7	88
SC Time	3	2	67	2	67
SC Space/ Time	1	1	100	1	100
AC in Space	4	3	75	2	50
AC in Time	4	3	75	3	75
Total	**20**	**16**	**80**	**15**	**75**

Table A-70: Engagement 4 ISU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	12	10	**83**	9	**75**
Not Vulnerable	8	6	**75**	6	**75**
Yes Answer	12	10	**83**	9	**75**
No Answer	8	6	**75**	6	**75**

A.4.4.2 Iowa State Specific Responses

A.4.4.2.1 Airplan 2

A.4.4.2.1.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Response: "airplan_2 uses a sort algorithm implementation (Orderer.rank) that has an AC in time vulnerability. The Orderer.rank sort implementation consumes excessive time when the number of items to sort is a multiple of 6 (see method Orderer.changingRank). This is due to a differential branch in the algorithm. […]"

Evaluation: Iowa identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.1.2 Question 038 (SC Space, Intended Vulnerable, Answered Yes

Response: "The number of airports in a routemap can be determined by requesting for the Passenger Capacity Matrix after uploading the routemap. The response for the passenger capacity matrix is an html page which contains a nXn matrix, where 'n' is the number of airports. The size of each cell of this matrix is constant, and because of this it is possible to deduce 'n', given the size of the html http response within the budget as follows. First, we could observe the response for the passenger capacity matrix, when n=1 as a passive operation. From that response, we can figure out what part of response remains the same and what part grows as n increases. The second active operation would be to get the passenger capacity matrix for a route map that the user is uploading. This will be the third response received after the user uploads the map. From the size of that response we can figure out what the size of airports is. [...] During the audit of FlightMatrixManger class, it was observed that a matrix of airports was returned where a CellFormatter was used to pad each cell in the output matrix to a fixed length. We hypothesized that this could lead to a side channel if the padding was constant. By auditing the CellFormatter's format method, we found that it takes 4 parameters (String s, int len, Justification j, boolean adjust). The second parameter is the final String length, which was set to 10 in this case. The last parameter "adjust" is important. If that is set to true, then random spaces are added to the String. In this case, this parameter was set to "false". So, each cell in the Matrix would be of size 10. This helped confirm our hypothesis that the number of airports (n) can be deduced from obtaining the size of the nXn matrix. The audit of FlightMatrixManager also revealed that the total response consists of a standard template. At first glance, the only 2 things that are variable in the response are: The airline name; The matrix

However, further analysis revealed that the airline name is padded to a fixed length. So, the only variable entity in the response is the matrix. [...]"

Evaluation: Iowa correctly determined the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.2 Airplan 3

A.4.4.2.2.1 Question 002 (SC Space, Intended Vulnerable, Answered Yes)

Response: "[...] For a side channel in space to leak the secret, we had to find an interaction, where the size of the response depends on the strongly connectedness property of the uploaded routemap. We used the reverse Call graph from AdjacencyListGraph.isConnected and forward data flow from the boolean returned by AdjacencyListGraph.isConnected to find out where the app uses this secret. We generated call graphs between the isConnected method and the entry points in the app, which showed that the method returning the secret is indeed reachable on a call sequence from an entry point. [...] To identify the observables related to the secret, we performed a data flow analysis using RULER. This led us to identify that if the routemap is is strongly connected, method MapPropertiesCoach.format takes a path that leaves the padded size at 19, otherwise, it resets it to 19*2 = 38. We use this branch in MapPropertiesCoach.format to form the side channel hypothesis. [...]"

Evaluation: Iowa identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.2.2 Question 016 (SC Space, Intended Not Vulnerable, Answered No)

Response: "The secret here is the number of airports. To identify a side channel in space that reveals the number of airports, we look for interactions with the app where the size of the response is only a function of the number of airports, and the additional data in the response (like flight information) remains constant on all responses. These responses would correspond to observables using which an attacker can deduce the number of airports using a side channel in space. The takeAirport method in RouteMap class was treated as a "secret" by our tool. The handleXXX methods in AbstractHttpCoach are the entry points through which the app interacts with the users (data is sent over the network). Our tool (IPCG's and call graphs) helped us audit the cases where the secret was passed over the network. […] We found 2 potential places where number of airports can be passed

1. In FlightMatrixCoach class which displays the passenger matrix

2. ViewRouteMapCoach class that displays the route map.

FlightMatrixCoach

The one potential place where the number of airports can be determined is by requesting the Passenger Capacity Matrix after uploading the routemap. The response for the passenger capacity matrix is an html page which contains a nXn matrix where 'n' is the number of airports. If the size of a cell in this matrix is constant then it could be possible to deduce 'n',, given the size of the html http response. During the audit of FlightMatrixCoach class, it was observed that a matrix of airports was returned using a CellFormatter. By auditing the CellFormatter's format method, we found that it takes 4 parameters (String s, int len, Justification j, boolean adjust). The second parameter is the final String length and which was set to 30 in this case. The last parameter "adjust" is important. If that is set to true, then random spaces are added to the String. In this case, this parameter was set to "true". So, each cell in the Matrix would not be of the same

size. After running an analysis on the bounds of what each cell size can be, we concluded that the size is random enough that we cannot conclude the number of airports from the Matrix size.

ViewRouteMapCoach

The ViewRouteMapCoach displays for each airport, the list of outgoing flights and the information for each flight, and can be a potential side channel that can reveal the number of airports based on size of html response, if the other response is constant. Since the number of outgoing flights can be different for different airports, the html generated for each airport is not fixed size. Hence it is not possible to guess the number of airports by looking at the whole html for all the airports."

Evaluation: Iowa correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.3 Airplan 4

A.4.4.2.3.1 Question 055 (SC Space, Intended Not Vulnerable, Answered No)

Response: Iowa repeated the analysis for Airplan 3 question 002 and concluded that the application is not vulnerable.

Evaluation: Iowa correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.4 Airplan 5

A.4.4.2.4.1 Question 005 (SC Space, Intended Not Vulnerable, Answered No)

Response: Iowa repeated the analysis for Airplan 2 question 038 and Airplan 3 question 16 and concluded that the application is not vulnerable.

Evaluation: Iowa correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.5 Malware Analyzer

A.4.4.2.5.1 Question 023 (AC Space, Intended Vulnerable, Answered No)

Response: "[…] The only potentially vulnerable code would be the application's interactions with its Logger.write_log_entry method. […] The only reference that is passed to the write_log_entry (as determined by dataflow analysis) is the Server.response global variable. The points-to analysis indicated that the Server.response variable is not aliased and only direct assignments to Server.response are used to set the value. Using data flow, the assignments to the "response" string are made in several locations that are all restricted to the Server class. Of these locations, there are 8 assignments made within loops and 5 assignments outside of loops (all assignments are within the Server.serve method and all assignments concatenate results to the existing string value). […] We determined the application was not vulnerable to an AC attack through an exhaustive investigation of each "response" variable assignment. For each assignment, we performed a backwards reachability analysis of the control and data flow to determine the worst-case input (in terms of the length of the string assigned to the "response" variable). After enumerating all worst-case inputs, we determined that the strength of the potentially vulnerable code was not strong enough to exceed the budget of 80KB. […]"

Evaluation: The challenge contained an intended vulnerability. An integer overflow vulnerability can be used to trigger a divide-by-zero exception. The resulting stack trace is then written to disk.

Post-Engagement Analysis Ruling: Incorrect

A.4.4.2.5.2 Question 058 (AC Time, Unintended Vulnerable, Answered Yes)

Response: "There is a simple attack that causes a denial of service by terminating the execution of the server. According to the operational definition of a Remote-DoS AC Time Vulnerability a 'remote challenge program (server or peer-to-peer) contains an AC Time vulnerability if and only if it is possible for an adversary to cause a benign user's request to exceed the resource usage limit while remaining within the input budget.' […] There is are multiple calls to System.exit in error handlers in the Logger class. This situation can be caused by sending requests simultaneously or in rapid succession, which causes a file I/O exception that is raised when the two server threads both access the "response.log" file. This error could occur in the exception handlers of Logger.initialize, Logger.write_log_entry, or the Logger.close_log method, each of which call System.exit terminating the server's execution. […]"

Evaluation: The challenge does not contain an intended vulnerability. Iowa referenced the OpDef v08.0 definition of an AC Time vulnerability. As stated in the documentation this is for E5 and as this is E4 does not apply. In either case the vulnerability identified by Iowa does not consume resources instead causing the challenge application to crash. As this fits the definition established by the question this response is noted as an out of scope unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.6 PowerBroker 1

A.4.4.2.6.1 Question 019 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Our analysis (detailed below) lead us to determine that several logging call sites are not guarded by a check to truncate long auction profile IDs. An attacker can use the majority of his 800,000 byte budget to announce a long auction profile ID which will then be written to the log. If the scenario is such that the attacker is the auction seller and the attacker wins his own bid our dynamic analysis revealed that the long profile ID will be written to the log multiple times allowing the 1,000,000 byte limit of the log file to be exceeded. […]"

Evaluation: The challenge contains an intended vulnerability in the connection establishment procedure. If an attacker disconnects from a user before the connection is fully established, the user's application will continually attempt to re-connect, logging each failed attempt. Iowa State identified the same unintended vulnerability as Draper and Northeastern.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.7 PowerBroker 2

A.4.4.2.7.1 Question 015 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "Powerbroker_2 has a protocol to conduct auctions and declare winner. The process to declare winner is as follows,

1. Everyone receives an encrypted packet which has information on everyone's bid value.

2. Every user then queries using this packet, whether his bid is the highest or not. Depending upon the answer he gets, he sends a winning claim or asks for concession (the message packet type is same).

3. This check is done using an unnecessarily complicated algorithm which compares two values byte by byte. (OfferAnalysisData.generate method)

4. This algorithm creates a singularity when the bid value is $300 and highest bid (winning bid) is $450. At this point of time, we observed a distinct spike.

Thus, attacker can detect another user's bid when the scenario leads to an auction where a user will bid within $30 of 300$ and the highest bid is around $450. […]"

Evaluation: The challenge does not contain an intended vulnerability. Iowa places constraints on the existence of the vulnerability that are not offered by the challenge question. Given the worst-case secret (as specified by the STAC operational definitions), this side channel is not believed to exist.

Post-Engagement Analysis Ruling: Incorrect

A.4.4.2.7.2 Question 025 (AC Space, Unintended Vulnerable, Answered Yes)

Response: "Our analysis (detailed below) lead us to determine that several logging call sites are not guarded by a check to truncate long auction profile IDs. An attacker can use the majority of his 800,000 byte budget to announce a long auction profile ID which will then be written to the log. If the scenario is such that the attacker is the auction seller and the attacker wins his own bid our dynamic analysis revealed that the long profile ID will be written to the log multiple times allowing the 1,000,000 byte limit of the log file to be exceeded. […]"

Evaluation: The challenge does not contain an intended vulnerability. Iowa State identified the same unintended vulnerability as Northeastern.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.8 SmartMail

A.4.4.2.8.1 Question 047 (SC Time/Space, Intended Vulnerable, Answered Yes)

Response: "The emails are iterated over in alphabetical order. The observables indicate where an administrator email occurs in the ordering of all involved emails. Knowing the public emails allows an attacker to infer that the private administrator is alphabetically between two known email addresses. […] By monitoring file sizes of the two logs over time, an attacker can determine how many times the regular log grows before the secure log grows to determine how many user accounts are before (and after) the administrator account. Using this information, we can binary search the administrator account one character at a time through a segmented side channel."

Evaluation: Iowa identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.8.2 Question 059 (AC Time, Unintended Vulnerable, Answered No)

Response: "The app has total of 5 control flow entry points, of which 4 are main methods, and 1 is a HTTP request handler, namely EmailSenderReceiver.doPost. We begin our analysis at this method because this is the entry point for client input according to the app description. […]

There are a total of 41 loops in the app. Using the call graph from the entry point, we identified the 3 methods called by the entry point that process the 3 kinds of input requests: (1) Sending email (EmailSenderReceiver.doEmail), (2) Receiving email (EmailSenderReceiver.getMbox) and (3) Look up email addresses in a mailing list (EmailSenderReceiver.getAddress).

We use LCG and loop reachability filter to identify loops executed when processing these 3 client interactions. The LCG for each of the above methods showed that the doEmail does the most processing in terms of number of loops (30). The other 2 contained only 5 and 10 loops (some loops overlap between the 3 functionalities, so the sum is greater than the 41 loops in the app). We audited these other 2 functionalities. They seemed relatively simple and did not lead to any AC Time hypothesis. So we take up doEmail for further analysis. […]"

Evaluation: Northeastern and Utah identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.4.2.9 Tour Planner

A.4.4.2.9.1 Question 056 (AC Time, Intended Not Vulnerable, Answered No)

Response: "There are 2 entry points corresponding to the 2 interactions allowed according to the app description: TourServlet (returns a tour consisting of cities provided in the input) and PlacesServlet (returns the list of places in the database). There was only 1 loop reachable (via control flow, identified using LCG) from PlacesServlet.doGet entry point. This loop iterates through the list of places (already populated during server start up). It was audited and did not lead to any AC Time vulnerability hypothesis. There were 13 loops reachable (via control flow, identified using LCG) from TourServlet.doGet entry point. […] By auditing these loops, the code computing the tour was identified to be in the TourCalcuator.calcMinSpanningTree method. This code also was responsible for issuing the progress reports back to the client after each city is found in the tour. […] we investigated the loops in the potentially vulnerable code - calcMinSpanningTree method. It was observed that the total number of cities is 25. […] The outer loop ensures that the tour computation runs only until the number of cities specified in the input has been added to the result. There is an inner nested loop that iterates over a sorted set of the edges between all pairs of cities (sorted lowest edge weight to highest edge weight), adds cities to the result tour, and reports progress to client via a HTTP message when each city is added. Thus, these 2 loops are the core components in constructing the tour. There are several conditions that are checked before the progress report is sent which were audited and they seemed routine for the algorithm. Since the database for the app consists of all pairs of cities (25 choose 2 = 300 edges), the loops will add at least one edge after iterating over all the 300. As a result, both the loops will eventually terminate. The worst-case number of iterations for the inner loop is 300 (total number of edges iterated before a city is added to a tour), and the outer loop runs the size of the input city list provided by the user. We verified that duplicate cities provided as input were removed before the calcMinSpanningTree method is invoked, and therefore the outer loop runs a maximum of 25 times. That is, maximum iteration of the edges is 25*300 = 7500 times per user query."

Evaluation: Iowa correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.10 Tweeter

A.4.4.2.10.1 Question 021 (AC Space, Intended Vulnerable, Answered Yes)

Response: "A tweet can be made with a post to /tweet and is received by TweetingController.tweet method that eventually passes the tweet to the SpellingCorrectionService.correctAndSaveTweet method, which contains an AC space vulnerability. […] After authenticating the user, the tweet input is passed to the SpellingCorrectionService.correctAndSaveTweet method. The correctAndSaveTweet method creates a background task to correct the spelling of the tweet. You can post to the back to endpoint, but you have to submit the equivalent of the following form (filling in the correct _csrf token). […] There are no bounds on the size of the tweet inside the application, so our input could be the maximum size of our allowed budget. There is however a server side request limit in the configuration of the Apache libraries running on the reference platform that practically limits the request to 1048576 bytes (this was found experimentally when trying to send longer strings equal to the maximum budget size). […] The maximum length of a word in a tweet is the maximum length of the word in the dictionary as calculated by Spelling.loadDictionary (the longest word in the dictionary). Quickly browsing the dictionary reveals "resourcefulness" as a candidate for longest word. An input consisting of the "resourcefulness" word with the typo in the last character position (e.g. resourcefulnesd") causes a high number of string operations because the edit distance window size is only 2 characters wide. Providing an input of "resourcefulnesd " concatenated repeatedly (156,249 times to stay within the overall attack budget or 65,535 to stay within the attack budget and the accepted apache request limit) exceeds the memory budget for this application. […] When we supplied a tweet of 156,249 resourcefulnesd word concatenations to the SpellingCorrectionService.correctAndSaveTweet method we observed a memory usage of 10,610,228 kilobytes. When we supplied a tweet of 65,535 resourcefulnesd word concatenations (to stay within the practical request size limit enforced by the Apache libraries) to the SpellingCorrectionService.correctAndSaveTweet method we observed a memory usage of 10,454,880 kilobytes. Both of which exceed the 1,500,000 kilobyte budget. This vulnerability was found by searching for deeply nested loops, filtering by loop monotonicity, and tainting with user controlled inputs. The Loop Call Graph for the methods of interest is shown in Figure 1. Inspecting those methods led to the testable hypothesis described above."

Evaluation: The application contains an intended vulnerability in the profile image size. The server allows users to select three difference sizes for profile images, and an attacker can use the budget to create several users with the largest profile image size. Iowa identified an unintended vulnerability that was addressed in the engagement 4 variant of the challenge program. As a result, the application is no longer believed to be vulnerable.

Post-Engagement Analysis Ruling: Incorrect

A.4.4.2.11 WithMi 1

A.4.4.2.11.1 Question 032 (SC Space, Intended Vulnerable, Answered Yes)

Response: "Whenever the users get added to a chat they need to update their copy of the HashSet WithmiChat.members. Once this is done as part of the protocol, users need to acknowledge the completion of the adding process via network. We hypothesize the number of acknowledgement packets generated to be a function of number of users and as attacker can observe the packets generated, he can figure out the number of users in a chat. […] We

hypothesized that the number of acknowledgement packets generated to be a function of number of users and as attacker can observe the packets generated, he can figure out the number of users in a chat. This requires to find the function f(n), where n is number of users in a chat, if it exists.

As there can be multiple members in a chat, it is likely that the packets are generated in a loop. We discovered all the loops which access hashset WithmiChat.members. For this, we used loop reachability filter to first identify the loops accessing the collection which stores information on members in a chat. We found 6 such loops. […] We examined these loops further. Loop in deliverMessage creates a serialized packet to be sent over the network for every other user in the chat. (See PCG) In other words, every user generates n packets if there are n users in a chat. The other loop then sends all the packets generated for every user. These loops are called one after the other. AddMemberCommand.execute calls the method containing these loops (see the callgraph in image below.) and PCG shows they are called one after other. Thus, we see n packets of same size $= f(n) = n + n = 2n$. […]"

Evaluation: Iowa identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.12 WithMi 2

A.4.4.2.12.1 Question 012 (SC Space, Intended Not Vulnerable, Answered No)

Response: "We can spend one of our budgeted operations to observe packets generated and the timings of each packet. Our static analysis revealed that the important packet contents are encrypted. Within our allowed passive operations we could potentially observe side channels due to the number of packets sent, the timings between packets, and any responses the receiving peer may send. While the file sizes of each file are very similar the length of the filenames are each unique. Experimentally, however, we found that the single response packet is always a fixed 61 byte response. This leaves only the sender's packet sizes and timings as observable side channels that are within scope for this question. […] Our analysis revealed that files can be processed in two ways. In both ways files are randomly padded, chunked, and then compressed before they are sent. The "sendfile" command corresponds to the "sneaker compression" routine and the "sendfilezlib" command uses the zlib compression library to compress the file. The chunks of the files are flushed out incrementally over the network. Both send file commands have an unencrypted message protocol prefix that can be used to identify the start of a file transfer (this byte value can be observed by the attacker listening with a packet capture on the victim's host machine; for example 0x0000017d corresponds to the "sendfile" command). Both routines flush raw data to the org.digitaltip.dialogs.TalkersConnection.write function. We instrumented this function for dynamic analysis to record the number calls and bytes written per call for a given file transfer. The results for each command are recorded in the tables below for 5 file transfer of each file and compression strategy. TalkersCryptoState initializes the cipher as this.encryptCipher = Cipher.getInstance("AES/CTR/NoPadding");. The cipher itself is unpadded, but there is an additional 48 byte overhead in byte array to be sent that comes from a file integrity hash (initialized as this.hmac = Mac.getInstance("HmacSHA256");) and then base64 encoded. Analysis of the padding strategy revealed that each chunk can be padded randomly with up to at most 256 bytes. FileTransfer.transmit is responsible for the precompression random padding. […] While it is unlikely that a single user session would consist of sending the same files multiple times, this experiment does reveal the challenge that randomized padding has added with our strict budget. In both transfer operations it is difficult to

to distinguish between the twinkle.txt and twinkle2.txt files. The ipsum.txt is also close in the average number of bytes and total size, but the observed variance in ranges is very high. The sendfile transfer operation is perhaps the most promising because the total number of bytes is significantly larger (suggesting a less optimized compression algorithm) than the sendfilezlib transfer operation. To examine this issue deeper, we increased the experimental data set to 50,000 samples for each file transfer of the sendfile operation. […] Sampling more data revealed that the variance is much too high. Even if a file was sent multiple times in a single session and we could use our remaining oracle query it would not be enough to differentiate between the ipsum.txt, twinkle.txt, and twinkle2.txt since our oracle query would only rule out one of the three potentially sent files. Of course with this data we could identify whether or not pi.txt or abc.txt was sent, but we must assume that any of the 5 files could be sent. This side channel is not strong enough to guarantee an attack with a 99% success rate."

Evaluation: Iowa correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.13 WithMi 4

A.4.4.2.13.1 Question 031 (SC Time, Intended Vulnerable, Answered Yes)

Response: Iowa performed the same analysis as for WithMi 5 question 027 and concluded that the challenge application was vulnerable.

Evaluation: Iowa identified the intended vulnerability and unintentionally switch the responses for WithMi 4 question 031 and WithMi 5 question 027. This mistake has been noted and is corrected in this report and in their score.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.14 WithMi 5

A.4.4.2.14.1 Question 027 (SC Time, Intended Not Vulnerable, Answered No)

Response: "Withmi maintains a history of the previous connections and it is stored in a file 'previous_users.txt'. At the start of a new session, contents of this file are read and are stored in a List - MemberManager.pastMembers. Whenever a new user is added to the chat, it is connected to the users already present in the chat. At this point it is stored in the aforementioned list as a Member. We hypothesized that at this point depending upon whether they are previously connected or not, this process will create a timing differential. […] Since we are interested in information about connection between two users, the entry point for the app is set to the Call Handler - NewConnectionCoach.call. This method will read the necessary information from channel and then create a workflow which the rest of the app uses. Thus, we are interested in callees of its child - HangIn.HandleConnection. Also note the presence of WithMiConnectionsService.readInPreviousConnections. This is where previous_users.txt is read. Note that after a member is added to addMemberToMemberHistory, no acknowledgement packet is sent. […] the only method which is of interest to us is ConversationManager.handleConferenceStateMessage and the call chains that originate from it. Thus, we hypothesize that when a chat is being formed this path will create a side channel depending upon whether the users were connected to each other previously or not. With this hypothesis in mind we explored further. […] ConversationManager.handleConferenceStateMessage has two loops in it. One adds all public

ids of members in a chat to a list which can later be used to send messages. Other loop ensures that members of a chat are connected to each other. […] PCG shows that if the users are previously connected then a check takes place which has an interprocedural call chain to connect - which will connect the two users. Now if they don't know each other then, the handleConnection route needs to be followed. It will form a new Member and add to a user's list of previously connected members, a loop which will add to the time difference, but since no packet is sent corresponding to the process. Attacker can't observe it. Hence, this side channel, although potentially strong enough to leak information, is not observable for attacker.

[…] Since, the only path with reachable loops was the one we examined in our first hypothesis, there is no singularity causing loop in this app which is related to the secret. Thus, we decided to analyze branches for the same. In the part of the app which is relevant to the secret there are 371 branch conditions. Out of them 106 are null checks, which can be ignored. 28 of them are instanceOf checks which can also be ignored. 80 of them are error code checks (== 0 or == -1 etc.). Rest of the conditions are different types of error checks such as if connection thread exists or a chat exists etc. The interesting one are when a collection is checked for having a value. This is done using containsKey API. The possibility we were after was if for a particular id this check has some expensive operation going on, i.e. if pastMembers does not have the key (not previously connected) do something expensive. There are 9 such conditions. We examined them manually. There was no expensive operation going on. Lastly, we reconfirmed our observations through dynamic analysis, that there is indeed no noticeable time difference. We created two different chats, one where the users were previously connected and in others they weren't. In both the time taken to generate packets fluctuates between 5 to 6 ms."

Evaluation: Iowa correctly concluded that the challenge does not contain an intended vulnerability and unintentionally switch the responses for WithMi 4 question 031 and WithMi 5 question 027. This mistake has been noted and is corrected in this report and in their score.

Post-Engagement Analysis Ruling: Correct

A.4.4.2.14.2 Question 046 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "[…] Whenever the users get added to a chat they need to update their copy of the HashSet WithmiChat.members. Once this is done as part of the protocol, users need to acknowledge the completion of the adding process via network. We hypothesize the number of acknowledgement packets generated to be a function of number of users and as attacker can observe the packets generated, he can figure out the number of users in a chat. […]"

Evaluation: Iowa reported the same potential unintended vulnerability as Vanderbilt and GrammaTech. The challenge question does not place further restrictions on the chats of the victim user. As a result, the user could be engaged in chats with two other users, and the number of chat messages would not be sufficient to disambiguate whether the user is in one chat with the two other users or in separate chats with each user.

Post-Engagement Analysis Ruling: Incorrect

A.4.5 Northeastern University

A.4.5.1 Northeastern Overview

Northeastern answered 51 questions with a 37% accuracy in E4. During E3, Northeastern answered the same number of questions but with a 67% accuracy. The team answered 41 more

questions in E4 than in E2 but had a 43 percentage point decrease in accuracy for E4. Northeastern identified unintended vulnerabilities in E3 that impacted 15 of their responses; however, as many of these were removed in the E4 variants, the team identified 3 unintended vulnerabilities for E4 that impacted only 3 questions. In effect, the E3 challenge programs contained vulnerabilities that Northeastern's tools were able to identify, but in E4 with a number of these removed as unintended vulnerabilities, their tools were unable to identify the remaining vulnerabilities.

Table A-71: Engagment 4 NEU Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	10	7	70	6	60
SC Time	12	6	50	4	33
SC Space/ Time	0	0	NA	0	NA
AC in Space	13	6	46	6	46
AC in Time	16	7	44	3	19
Total	**51**	**26**	**51**	**19**	**37**

Table A-72: Engagement 4 NEU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	32	16	**50**	9	**28**
Not Vulnerable	19	10	**53**	10	**53**
Yes Answer	24	16	**67**	9	**38**
No Answer	26	10	**38**	10	**38**

A.4.5.2 Northeastern Specific Responses

A.4.5.2.1 Airplan 1

A.4.5.2.1.1 Question 020 (AC Time, Intended Vulnerable, Answered No)

Response: "No vulnerabilities were found."

Evaluation: Northeastern did not observe that the shortest path algorithm is a vulnerable A* algorithm with an inconsistent heuristic.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.1.2 Question 051 (AC Space, Unintended Vulnerable, Answered No)

Response: "No vulnerabilities were found."

Evaluation: Draper and UC Boulder identified unintended vulnerabilities in the challenge application.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.2 Airplan 2

A.4.5.2.2.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Response: "The max-flow algorithm used is the Ford-Fulkerson algorithm which has a very bad worst case where the run time is proportional to the largest individual flow."

Evaluation: This variant of the challenge uses a non-vulnerable implementation of Ford-Fulkerson. We do not believe that the application is vulnerable to this reported vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.2.2 Question 038 (SC Space, Intended Vulnerable, Answered Yes)

Response: "The response of matrix request is $O(n^2)$ bytes in number of airports, so an attacker that observes this encrypted message can infer the number of airports reliably. The tool flagged the influence of secret data on the construction of the matrix and its transmission on a network channel. The length of each cell is fixed regardless of the data stored in the route map. This gives a reliable side-channel for an adversary to recover the number of airports; the size of the capacity matrix response uniquely identifies how many airports are stored in the routemap."

Evaluation: Northeastern identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.3 Airplan 3

A.4.5.2.3.1 Question 002 (SC Space, Intended Vulnerable, Answered Yes)

Response: "We made a proof of concept that shows how an adversary could determine whether the route map was strongly connected by just viewing the size of packets in a response. We didn't define what the worst-case routemap would be for this approach."

Evaluation: Northeastern identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.3.2 Question 016 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "The question is a little vague. The program is vulnerable depending on what is considered as ""active operation"". We first uploaded an empty routemap to get the size of an empty routemap. Then, by adding an airport to the map, we can determine the padding size. Assuming the attacker can upload any number of routemaps in his own session, he can generate a single variate regression function that predicts the number of airports based on the size of the response. In this approach, the active operation is the "find a capacity" in "https://127.0.0.1:8443/route_map/id", where id is a number associated with the uploaded routemap, after the user loads the routemap. We record the network traffic from uploading the routemap. Regardless of the number of connections between airports and how strongly connected they are, this response size is fixed and depends on the number of airports in the routemap. A POC to analyse a pcap, and a sample pcap file are available."

Evaluation: Random whitespace padding is applied to prevent a mapping of number of airports to observed packet size. Further testing by the AC team responsible for the challenge confirmed the lack of a vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.3.3 Question 024 (AC Time, Intended Vulnerable, Answered Yes)

Response: "Found a complexity attack in TextFileLoader, the number of airports is not checked until after loading, so an [attacker] can provide a file with 3800 airports. For each airport it will run getAirport() which has an expensive loop over all the previous airports and will exceed the allotted time. The loop was one of the list flagged as potentially expensive given its dependence on attacker input and control flow structure."

Evaluation: The intended vulnerability was due to a vulnerable Ford-Fulkerson implementation. The potential vulnerability identified by Northeastern is one that was reported and fixed from E3. The number of airports in check prior to loading, and inputs crafted to demonstrate the reported vulnerability terminated in less than 6 seconds with an error message.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.4 Airplan 4

A.4.5.2.4.1 Question 022 (AC Time, Intended Vulnerable, Answered No)

Response: "No vulnerabilities were found."

Evaluation: Northeastern did not observe that the shortest path algorithm is a vulnerable Dijkstra's implementation.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.4.2 Question 055 (SC Space, Intended Not Vulnerable, Answered No)

Response: "The tool reported the influence of secret data on attacker observables. However, the properties page has the same size whether or not it is fully connected so there is no side channel in space that can allow an attacker to determine if the graph is fully connected."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.5 Airplan 5

A.4.5.2.5.1 Question 005 (SC Space, Intended Not Vulnerable, Answered No)

Response: "The response of the matrix request is $O(n^2)$ bytes in number of airports, so an attacker that observes this encrypted message may infer the number of airports. The tool flagged the influence of secret data on the construction of the matrix and its transmission on a network channel. The length of each cell in the capacity matrix is derived by the cell's corresponding data in the routemap. In the worst-case, an adversary would not be able to recover the number of airports in only 2 operations due to the variation of each cell's length."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.6 BidPal 1

A.4.5.2.6.1 Question 004 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "MgMultiplier.exponentiate() has a timing difference between paths that are controlled by individual bits of D. However, I do not have enough time to PoC the attack."

Evaluation: The communications framework of the application coupled with the small budget is believed to prevent this reported vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.6.2 Question 033 (AC Time, Unintended Vulnerable, Answered No)

Response: "No vulnerability was found."

Evaluation: Draper, Vanderbilt, and Invincea discovered unintended vulnerabilities.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.7 BidPal 2

A.4.5.2.7.1 Question 044 (AC Time, Intended Vulnerable, Answered No)

Response: "No vulnerability was found."

Evaluation: Northeastern did not observe that provided a particular username and auction ID, the application continuously calculates a checksum which fails. Each failed checksum is logged to a file on disk.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.7.2 Question 057 (SC Time, Intended Vulnerable, Answered Yes)

Response: "MgMultiplier.exponentiate() has a timing difference between paths that are controlled by individual bits of D. This can be used to leak another user's bid."

Evaluation: The challenge application contains a vulnerability in bid comparison that results in a user's bid being leaked. Northeastern's response does not indicate that they identified the intended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.8 Collab

A.4.5.2.8.1 Question 028 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "The tool reported that untrusted input flows to methods that manipulate a B-Tree structure. This program queries/updates this B-Tree structure every time events are searched/added. The time needed to update the B-Tree is not constant since, sometimes, the function "split" is called (when the bucket of values associated with a node of the tree gets full). An attacker can measure the time needed to add an event to understand when a split operation in the tree happened. Empirically, we noticed that the insertion time is sometimes longer and we verified (by using a debugger) that the cause of this slowdown is indeed the execution of the function: collab.SchedulingSandbox.split(DataNode orignode). An attacker can also query (using the "search" operation) all the events it is suppose to see and compute when a specific insertion is suppose to trigger a split operation. If the attacker's prediction is not correct, this means that an extra element is present in the tree in the node modified by the last insertion. This extra element must necessary be an audit event. Therefore, an attacker can progressively narrow-down the location in the tree where an audit event is stored. In turn, this allows an attacker to logarithmically (within the allowed input budget) narrow-down the range of values the audit event has. Finally, an attacker can query a hypothesized value for an audit event by trying to insert an event with the same id and check whether the error: "DuplicateEvent" is returned."

Evaluation: The challenge application non-locally balances the non-split cases such that the timing distribution of a split vs non-split insert operations are sufficiently overlapped. The challenge application as a result is not believed to be vulnerable.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.8.2 Question 035 (AC Time, Intended Not Vulnerable, Answered No)

Response: "The tree used to search for events is kept balanced. We checked all the operations and all the identified loops and recursive calls. We think that all of them terminate within the allowed time."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.8.3 Question 049 (SC Space, Intended Not Vulnerable, Answered No)

Response: "The tool enumerated the points at which secret information is transmitted by the application. The application communicates using fixed-length fields and structures. Additionally, the number of packets returned by most commands is fixed. Finally, the commands returning a variable number of packets (e.g., SEARCH), do not reveal any secret information. Therefore, we think that there is no space side-channel vulnerability in this application."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.9 Info Trader

A.4.5.2.9.1 Question 008 (AC Time, Intended Not Vulnerable, Answered Yes)

Response: "The tool reported a recursive method invocation that it was not able to establish a base case for. Given the way in which the program processes links, a cycle in the hyperlink graph will lead to infinite recursion. This happens in the following line: infotrader.dataprocessing.SiteMapGenerator.serializeNode(SiteMapGenerator.java:404) A simple cycle (e.g., d1 links to d2 and d2 links to d1) generates a java.lang.StackOverflowError due to recursion before the 1000 second limit. However, it is possible to create a structure of links and documents exceeding this limit when processed. Specifically, the following input triggers the vulnerability: ./postdocument.sh d20.txt; ./postdocument.sh d19.txt; ./postdocument.sh d18.txt; ./postdocument.sh d17.txt; ./postdocument.sh d16.txt; ./postdocument.sh d15.txt; ./postdocument.sh d14.txt; ./postdocument.sh d13.txt; ./postdocument.sh d12.txt; ./postdocument.sh d11.txt; ./postdocument.sh d10.txt; ./postdocument.sh d9.txt; ./postdocument.sh d8.txt; ./postdocument.sh d7.txt; ./postdocument.sh d6.txt; ./postdocument.sh d5.txt; ./postdocument.sh d4.txt; ./postdocument.sh d3.txt; ./postdocument.sh d2_1.txt ; ./postdocument.sh d2_2.txt ; ./postdocument.sh d1.txt Where d20.txt contains a document named d20, d19.txt contains a document named d19 linking to d20, d18.txt contains a document named d18 linking to d19, and so on up to d2_1.txt. d2_1.txt is named d2 and links to a document named d1. d2_2.txt is named d2 and links to the document d3 AND d1. d1.txt is named d1 and links to d2."

Evaluation: Northeastern reported a potential vulnerability in the document cycling structure. This vulnerability was reported for E3 and was addressed in the E4 variant of the challenge.

After attempts by the AC team responsible for the challenge to re-create this vulnerability we believe the measures taken from E3 were sufficient to remove it.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.9.2 Question 042 (AC Space, Intended Vulnerable, Answered No)

Response: "Given how documents are processed, the only file that could significantly increase its size is the sitemap xml file (which was among those reported as potentially untrusted input by the tool). However, the content of this file only contains part of the document and a non-recursive list of links. In fact, if the link structure does not contain a cycle, it is impossible to hit the 100,000 KB threshold. (also because only the first link is followed when the tree is serialized). On the contrary, if the link structure contains a cycle, then the execution will not terminate (or it will terminate with a StackOverflowError) and, therefore, no sitemap file will be generated."

Evaluation: Northeastern missed that the challenge contained a broken input guard that could be used to create a complex looping structure.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.10 Linear Algebra Platform

A.4.5.2.10.1 Question 034 (AC Space, Intended Vulnerable, Answered No)

Response: "The tool reported the influence of untrusted input on each operation. Each functionality was tested with increasingly larger inputs and the space complexity was not large enough to exceed limits given the constraints on the input."

Evaluation: The challenge application allocates memory for the vector input to the shortest path operation before checking the input size. The amount of memory allocated is attacker controlled.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.10.2 Question 050 (AC Time, Intended Vulnerable, Answered Yes)

Response: "The tool reported the influence of untrusted input on the matrix muitiplication request. The parallelized matrix multiplication functionality has n^3 complexity, combine with using small floating point numbers (denormalized) ie 0000000019088e-40 and 650x650 matrices and it will take more than the allotted time. (Took 4min5sec for one test)"

Evaluation: The challenge application uses two threads to compute the matrix multiplication. The task distribution is attacker controlled. Northeastern reported a potential unintended vulnerability in E3 that was addressed for E4 by preventing scientific notation on user inputs. Within the input budget we do not believe this is vulnerable.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.11 Malware Analyzer

A.4.5.2.11.1 Question 023 (AC Space, Intended Vulnerable, Answered No)

Response: "The code in the DasmHelper.class is updated with a hashset (assigned_instructions) which keeps the address of the instruction as key. Before the program proceeds it checks whether the given address is already availble in the hashset. This patched the program in E4."

Evaluation: Northeastern did not observe that an integer overflow can be used to trigger a divide by zero exception, which in turn will cause a large stack trace to be written to a log file.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.11.2 Question 058 (AC Time, Unintended Vulnerable, Answered No)

Response: "No vulnerability was found."

Evaluation: Invincea identified an intended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.12 PowerBroker 1

A.4.5.2.12.1 Question 019 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Since the attacker controls his own ID, and there is no check against the size of the ID, and his ID will be written to the log file on the remote side multiple times, the attacker can perform this attack by having a really long ID in his own .id file. In our PoC, we managed to generate log files of more than 2 MB on every participant other than the attacker himself by having a PDU of around 400 KB. In this challenge, our static analysis tool fails to identify any file write (since really there is no FileWriter at all) from the control flow graph, which led us to incorrect believe there is no such vulnerability. Later a manual reversing and inspection showed that a log file is generated, and we managed to perform the attack."

Evaluation: The application contained an intended vulnerability where if an attacker connects to a user a disconnects before the connection is established, the user will continuously attempt to re-connect, logging each attempt to disk. Northeastern and Draper identified that the challenge application does not check ID length properly. As a result, a long ID is written to the log file resulting in increased disk usage.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.12.2 Question 040 (SC Time, Intended Vulnerable, Answered Yes)

Response: "BigNumber.compareTo() has a small timing difference between different paths, and there is no blinding. However, I'm not sure if it is exploitable within the budget."

Evaluation: The challenge program contains a vulnerability in the bid comparison that allows an attacker to map the response time to the user's bid. Northeastern flagged a potential vulnerability that is not believed to be sufficiently strong.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.13 PowerBroker 2

A.4.5.2.13.1 Question 015 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "BigNumber.compareTo() has a small timing difference between different paths, and there is no blinding. However, I'm not sure if it is exploitable within the budget."

Evaluation: The challenge does not contain an intended vulnerability. We do not believe the vulnerability identified is sufficiently strong.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.13.2 Question 025 (AC Space, Unintended Vulnerable, Answered Yes)

Response: "Since the attacker controls his own ID, and there is no check against the size of the ID, and his ID will be written to the log file on the remote side multiple times, the attacker can perform this attack by having a really long ID in his own .id file. In our PoC, we managed to generate log files of more than 2 MB on every participant other than the attacker himself by having a PDU of around 400 KB. In this challenge, our static analysis tool fails to identify any file write (since really there is no FileWriter at all) from the control flow graph, which led us to incorrect believe there is no such vulnerability. Later a manual reversing and inspection showed that a log file is generated, and we managed to perform the attack."

Evaluation: The challenge program does not contain an intended vulnerability. Northeastern and Draper identified that the challenge application does not check ID length properly. As a result, a long ID is written to the log file resulting in increased disk usage.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.14 PowerBroker 4

A.4.5.2.14.1 Question 006 (SC Time, Intended Not Vulnerable, Answered No)

Response: "No vulnerability was found."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.15 RSA Commander

A.4.5.2.15.1 Question 039 (AC Time, Intended Vulnerable, Answered Yes)

Response: "The tool reported that untrusted input influences a loop in client connection handling. Manual inspection revealed that the server code does not properly handle closed connections, therefore a malicious client can cause the server's code to enter in an infinite loop. The code locations where improper validation happens are in the Communication class, when the "handle" function of the objects handshakeHandler or requestHandler is called. Moreover, the function HANDLER_STATE handle(InputStream is, PacketBuffer packetBuffer) returns HANDLER_STATE.WAITING when the connection is interrupted (because the function isPackedStillOk always returns true). Therefore closing the "sending" side of a connection makes the server code enter an infinite loop. After this operation, the client will never receive any communication back from the server and a server's thread will enter an infinite loop. This behavior can be triggered, for instance, with Python code that first partially sends the first message of the handshake and then executes s.shutdown(socket.SHUT_WR) (where s is a socket instance connected to the server)."

Evaluation: The challenge contains a slow mode of RSA decryption where the attacker controls the runtime. The vulnerability identified by Northeastern is ruled out by the additional background for the challenge question.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.15.2 Question 053 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "The tool reported that secret data is used to decide the execution of paths with potentially inequal running times in its cryptographic routines. We found that this software uses the Java BigInteger implementation of modular exponentiation. We verified that this

implementation is not constant-time, specifically, on the reference platform, given two exponents e1 and e2, with e2 containing one more bit set to 1 than e1, the execution of (b^e2 mod p) takes about 0.05ms more than (b^e1 mod p).

An attacker can use this information to analyze the time needed by the application to encrypt a set of chosen plain text messages and recover the private exponent of the public key used during a previous conversation between the victim's client and the server. Knowing the private exponent, an attacker can recover the symmetric key exchanged during the victim's previous communication and therefore decrypt the conversation.

Please also note that the application uses weak encryption algorithms. Specifically, it uses by default a 512bit rsa key (which is factorable in a reasonable amount of time) and it does not allow users to use larger keys. Also, it uses DES for symmetric encryption, which has been proved to be vulnerable. However, we consider these security issues as outside the scope of this engagement."

Evaluation: The challenge program does not contain an intended SC Time vulnerability. Northeastern reported a vulnerability in the java.math.BigInteger implementation of modular exponentiation. The challenge program uses both the BigInteger.modPow() and the BigInteger.pow() and BigInteger.mod() implementations of modular exponentiation conditioned on the user input. Northeastern reported a timing difference of 0.05 ms depending on whether a given bit of two otherwise identical exponents is set (1) or not set (0). It is unclear how this information can be used to segment and test each bit of the exponent individually. Given the noise on the reference platform in the current network configuration and the operational budget of 10,000 operations, we do not believe that the challenge program is vulnerable.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.16 SmartMail

A.4.5.2.16.1 Question 043 (AC Space, Intended Not Vulnerable, Answered No)

Response: "None of the operations the mail server offers seems vulnerable to space complexity with the resource limit and budget provided."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.16.2 Question 059 (AC Time, Unintended Vulnerable, Answered Yes)

Response: "The challenge has an algorithmic complexity vulnerability. Every request sent to the server is processed sequentially. In the email.cgi operation where the user can send an email to other users, the program applies a parsing operation and a map reduce algorithm to the bytes of the email. In case the email processed has in the 'To' field a mailing list address (e.g. security@smartmail.com) this process takes more time. In case the same email contains multiple security@smartmail.com in the 'to' list of addresses this time further increase due to the algorithm design. Therefore, if multiple requests are sent to the email.cgi operation (up to fill the budget) crafted with a destination address list that contains multiple security@smartmail.com, the last request gets delayed of more than the resource usage limit provided. In our POC we crafted an email with 17 security@smartmail.com in the To field, no subj and "A" as content and sent it 17 times to the server (586bytes per request). The last request is always received in more than 30 seconds which is the resource usage limit."

Evaluation: The challenge program does not contain an intended vulnerability. Northeastern identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.17 Tour Planner

A.4.5.2.17.1 Question 011 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "Assuming a user's 5 desired cities are distinct, we can determine the unordered list of cities by observing both the request and response packet lengths. We experimentally noticed that encrypted traffic sent to and from the application is not being padded, meaning we can infer the total length of the request containing the list of cities, and the response containing the route. Given the list of cities, and their formatting as returned from the application, we can calculate the set of possible cities for a given request and response length, requiring only 2356 possible guesses in the worst case, which is less than the 3000 guesses allotted by the question."

Evaluation: The unordered set of user request result in 53,130 possible combinations and the application uses an AES256 cypher with a block size of 128 bits. The worst-case collision for all possible secrets is an observable encrypted output size of 405 bytes. When coupled with the encrypted request sizes, the worst-case pair of encrypted request and respond sizes was 218 bytes and 405 bytes, resulting in 24,696 collisions. This is not resolvable within 2999 oracle queries.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.17.2 Question 045 (SC Time, Intended Vulnerable, Answered No)

Response: "There is a clear timing side channel dependent on the geographical distance between the cities requested by the user. During testing, we found that with data collected on the reference platform a guess for a given tour would take ~1300-1500 guesses to the oracle to reach a success probability of 50%. For our example, the standard deviation of the distance between our guessed tour and the correct tour was 1272.8767800470587 so we would need around 2 * standard deviation = ~2500 guesses in both directions, so ~5000 guesses to achieve a confidence level of 95%. Therefore, getting an accuracy of 95% within the given operational budget of 300 operations seems impossible."

Evaluation: The challenge program contains a time side channel that allows an attacker to map each user request based on the collective set of response times from the application. An in-depth analysis is available in the Tour Planner whitepaper.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.17.3 Question 056 (AC Time, Intended Not Vulnerable, Answered No)

Response: "The tour implementation looks to be implemented so that no input could create that much of a delay since the only parts with either deep recursion or highly looping are controlled by information not controllable by the attacker. No requests to the /route application incur overhead that approaches the specified amount of time spent of 30 seconds. The most I can achieve is around 9 seconds by turning on GPX output and not encoding the points. Even then the time comes from the delay of writing the data to the network, not from the operations of the algorithm. By limiting the explorable region to only massachusetts and the request size to 500 with each &point parameter taking up about ~30 bytes there is not enough room to significantly impact time complexity. Because the configuration by default implements

ContractionHierarchies we can only set the algorithms to 'astarbi' and 'dijkstrabi' which does not allow for any significant overhead."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.18 Tweeter

A.4.5.2.18.1 Question 021 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Creating a user takes 893 bytes of PDU (incl. ethernet frame). Creating a user with avatar size=512 takes up >1MB (512x512 BMP). Therefore, by creating >1500 users we can reach >1500MB memory used within the budget of 2.5MB PDU."

Evaluation: Northeastern identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.18.2 Question 029 (AC Time, Intended Vulnerable, Answered No)

Response: "No request to Tweeter takes up significant timing resources meeting the minimum bar. The closest to being ACT is forcing as many hashtag or spelling corrections lookups as possible, however, it does not meet the bar. Similarily, forcing hashCode collisions on the hashtag set does not meet the required response of 1500 seconds."

Evaluation: A vulnerability in the spell correction feature can be used to trigger an AC Time vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.19 WithMi 1

A.4.5.2.19.1 Question 007 (AC Space, Intended Not Vulnerable, Answered No)

Response: "The problem is exposed during fuzzing stage: repeatedly sending files via command "sendfilezlib N" will take an excessive amount of memory which is never freed afterwards. However, we are not able to construct a POC less than 10,000 bytes to make the WithMI application use more than 2000 MB of memory. We do not think this application possesses the specified vulnerability."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.19.2 Question 017 (AC Time, Intended Vulnerable, Answered No)

Response: "We cannot find any path leading to triggering an algorithmic complexity vulnerability in time in this application."

Evaluation: The Huffman compression implementation contains an AC Time vulnerability expressed during de-compression.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.19.3 Question 032 (SC Space, Intended Vulnerable, Answered Yes)

Response: "We took the challenge and executed it together with tcpdump to monitor the traffic. When talking in a private chat, packets are only exchanged among all users participated in the

chat. Therefore, an attacker can easily figure out how many users are involved by observing how many IP addresses/ports there are in all from observing the traffic, which yields the number of participants of the private chat."

Evaluation: The challenge program contained an intended SC Space vulnerability: when a new user is added to a WithMi session, the user receives a chat message larger than all other WithMi messages. The size of this message is proportional to the number of users in the chat. Northeastern identified a potential unintended vulnerability which is to identify the number of unique IP address and port pairs and map this to the number of users in a chat. The challenge question does not restrict the victim user to a single chat. Thus, the pairing of host and port values is not sufficient to determine the number of users in each chat.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.20 WithMi 2

A.4.5.2.20.1 Question 018 (AC Space, Intended Vulnerable, Answered No)

Response: "The problem is exposed during fuzzing stage: repeatedly sending files via command "sendfilezlib N" will take an excessive amount of memory which is never freed afterwards. However, we are not able to construct a POC less than 10,000 bytes to make the WithMI application use more than 2000 MB of memory. We do not think this application possesses the specified vulnerability."

Evaluation: Northeastern did not identify the vulnerable Huffman Compression implementation that contains an AC Space vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.21 WithMi 3

A.4.5.2.21.1 Question 026 (SC Time, Intended Not Vulnerable, Answered No)

Response: "As pointed out in our path cost analysis, there should be a small timing difference between two paths inside the inner loop of MontgomeryMultiplier.exponentiate(). However, further manual inspection shows that the EncryptionPrivateKey.decrypt() function implements blinding securely, which makes it impossible to exploit the timing side-channel."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.21.2 Question 030 (AC Space, Intended Vulnerable, Answered Yes)

Response: "In CoolCompression.unzip(), num (which is returned by function readInt(stream)) can be a negative number, which essentially escapes the maximum size limit by the condition check of (totalSize > MAX_SIZE). An attacker can thus construct a special request that writes an excessive amount of data to the unzipped file. The loop was among the ones that are pointed out by our static loop analysis, and then we manually inspected it to confirm the finding."

Evaluation: Northeastern identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.22 WithMi 4

A.4.5.2.22.1 Question 010 (SC Space, Intended Vulnerable, Answered Yes)

Response: "By analyzing FileTransfer.send() function, we learned that file content is directly coming from FileInputStream(), and directly put into the compression implementation. Since compressing the five files provided with the challenge yields different sizes, we believe there is likely to be a side channel in space: by observing how many bytes are sent to the other user. We further verified it by sending different files in the controlled environment via file-sending commands, and found it was indeed the case."

Evaluation: Northeastern identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.22.2 Question 031 (SC Time, Intended Vulnerable, Answered Yes)

Response: "In ParticipantConductor.addParticipantToParticipantHistory() method, the variable previouslyConnected (together with another variable) dominates three paths, and static path cost analysis shows there should be a noticeable difference in execution time between two of them, which is because method storeParticipant() eventually goes into file access, while the other path with previouslyConnected == true only performs a write to System.out. Further manual investigation shows observations can be made at the time when the add participant message is received, and at the time a receipt is sent back (by calling sendReceipt() in addParticipantToParticipantHistory() method), and luckily, the difference in execution time can be reflected in the observation of time elapsed between the two events. Due to time constraint, we assumed that the difference of path costs is very observable and didn't bother to PoC this question."

Evaluation: Northeastern identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.22.3 Question 036 (AC Space, Intended Vulnerable, Answered No)

Response: "In AwesomeSmashing.unzip(), num has to be greater than 0 in order to conduct a write to output stream. The loop in AwesomeSmashing.unzip() shows up in static loop analysis, since our analysis could not resolve the number range of num. Manual inspection made us believe it is not vulnerable to space complexity attack in that function. Also, we manually analyzed other paths and concluded that no other path can lead to a file write."

Evaluation: Northeastern did not observe that the RLE compression implementation contains an AC Space vulnerability due to an integer overflow.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.22.4 Question 041 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "This one is almost the same as Question 026, but without blinding implemented. Theoretically it is possible to exploit the timing side-channel in mgMultiply() (the extra reduction)."

Evaluation: The Montgomery multiplication implementation in this variant of the challenge program is not believed to be vulnerable. Chinese Remainder Theorem is not implemented, and the communication framework is believed to effectively prevent leaking a sufficient number of bits.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.23 WithMi 5

A.4.5.2.23.1 Question 014 (AC Space, Unintended Vulnerable, Answered No)

Response: "We weren't able to use up the 2 GB of memory space within the given budget."

Evaluation: Invincea identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.23.2 Question 027 (SC Time, Intended Not Vulnerable, Answered No)

Response: "In ParticipantConductor.addParticipantToParticipantHistory() method, the variable previouslyConnected (together with another variable) dominates three paths, and static path cost analysis shows there should be a noticeable difference in execution time between two of them, which is because method addParticipantToParticipantHistoryUtility() eventually goes into file access, while the other path with previouslyConnected == true only performs a write to System.out. However, such a difference is not reflected in observation, since the receipt is sent prior to addParticipantToParticipantHistoryUtility() is called. The two different paths controlled by variable previouslyConnected do not have a significant execution time difference. Hence we believe the side channel is not exploitable (or rather, there is no timing side channel to an external observer)."

Evaluation: Northeastern correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.5.2.23.3 Question 046 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "We took the challenge and executed it together with tcpdump to monitor the traffic. When talking in a private chat, packets are only exchanged among all users participated in the chat. Therefore, an attacker can easily figure out how many users are involved by observing how many IP addresses/ports there are in all from observing the traffic, which yields the number of participants of the private chat."

Evaluation: The challenge program does not contain an intended SC Space vulnerability. Other variants of this challenge program leak information via the chat state messages. In this variant, padding is added to mask the number of users being leaked in the chat state messages. Northeastern identified a potential unintended vulnerability which is to identify the number of unique IP address and port pairs and map this to the number of users in a chat. The challenge question does not restrict the victim user to a single chat. Thus, the pairing of host and port values is not sufficient to determine the number of users in each chat.

Post-Engagement Analysis Ruling: Incorrect

A.4.5.2.24 WithMi 6

A.4.5.2.24.1 Question 054 (AC Time, Unintended Vulnerable, Answered No)

Response: "We weren't able to use up the 2 GB of memory space within the given budget."

Evaluation: GrammaTech and Invincea identified unintended vulnerabilities.

Post-Engagement Analysis Ruling: Incorrect

A.4.6 University of Maryland

A.4.6.1 UMD Overview

UMD answered 33 questions with a 61% accuracy. The team answered 21 more questions in E4 than in E2 and 22 more questions than in E1. Their accuracy for E4 was 3 percentage points less than their E2 accuracy but 3 percentage points more than their E3 accuracy.

Table A-73: Engagement 4 UMD Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	6	67	6	67
SC Time	10	7	70	6	60
SC Space/ Time	1	0	0	0	0
AC in Space	9	6	67	6	67
AC in Time	4	2	50	2	50
Total	**33**	**21**	**64**	**20**	**61**

Table A-74: Engagement 4 UMD Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	18	11	**61**	10	**56**
Not Vulnerable	15	10	**67**	10	**67**
Yes Answer	16	11	**69**	10	**63**
No Answer	17	10	**59**	10	**59**

A.4.6.2 UMD Specific Responses

A.4.6.2.1 Airplan 2

A.4.6.2.1.1 Question 038 (SC Space, Intended Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis, decompiler and jProfiler. There is a vulnerability, assuming that the victim user, after uploading a route map, will go through by clicking 'Next' in the following pages, and thus reach the one with path '/passenger_capacity_matrix'. The rows of the matrix generated correspond to the number of airports, and the formatting of those rows is using the method 'format' in FlightMatrixManager, which calls the method with the same name of the class CellFormatter. The values passed to the latter method call, in particular 10 for len and false for adjust, cause the length of those formatted strings to not depend on random variables. We believe it is thus possible for an attacker to determine the number of airports from observing the size of the HTML output."

Evaluation: UMD identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.2 Airplan 3

A.4.6.2.2.1 Question 002 (SC Space, Intended Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis, resource bound analysis (analyzed 1332 methods in 75 minutes), decompiler and jProfiler. We believe there is a vulnerability. We assume the victim user will continue clicking on 'Next' after uploading a route map, and thus reach the page with path '/map_properties'. The information about whether a graph is connected is in the string connectedString and obtained from calling schemeAdapter.obtainConnected() during the execution of grabContentsForPost in the class MapPropertiesGuide. This string is either "Fully Connected" if the graph is strongly connected or "Not Fully Connected" otherwise. A difference of four characters may not be enough to witness this in the HTML output. The format method, checks if the length of the string to be formatted is larger than this.valueLength, which is 19, and in the case of not being fully connected the latter is not the case. Therefore, the format of the class CellFormatter is called with a new length as input, being final int n = this.valueLength * (int)this.paramGuide.get("adjustment_factor"); which is 2*this.valueLength. We believe the difference of more than 19 characters allocated in the HTML output, should be enough to determine whether the graph is strongly connected."

Evaluation: UMD accidentally switched the responses to Airplan 3 Q 002 and Airplan 4 Q055, this is noted and corrected. UMD identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.2.2 Question 016 (SC Space, Intended Not Vulnerable, Answered No)

Response: "We believe there is no vulnerability even assuming the victim user will continue clicking on 'Next' after uploading a route map, and thus reach the page with path '/passenger_capacity_matrix'. The rows of the matrix generated correspond to the number of airports, and the formatting of those rows is using the method 'format' in FlightMatrixCoach, which calls the method with the same name of the class CellFormatter. The values passed to the latter method call, in particular 30 for len and true for adjust, cause the length of those formatted strings to depend on random variables and we thus believe it is difficult for an attacker to determine the number of airports from observing the size of the HTML output."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.3 Airplan 4

A.4.6.2.3.1 Question 055 (SC Space, Intended Not Vulnerable, Answered No)

Response: "Tools we used: static taint analysis, resource bound analysis (analyzed 1319 methods in 90 minutes), decompiler and jProfiler. We believe there is no vulnerability, even assuming the victim user will continue clicking on 'Next' after uploading a route map, and thus reach the page with path '/map_properties'. The information about whether a graph is connected is in the string connectedString and obtained from calling schemeAdapter.obtainConnected() during the execution of grabContentsForPost in the class MapPropertiesGuide. This string is either "Fully Connected" if the graph is strongly connected or "Not Fully Connected" otherwise. A difference of four characters may not be enough to witness this in the HTML output. The format method, checks if the length of the string to be formatted is larger than this.valueLength, which is 19, and in the case of not being fully connected the latter is not the case. Therefore, the

format of the class CellFormatter is called with a new length as input, being final int n = this.valueLength * (int)this.paramGuide.get("adjustment_factor"); which is still equal to this.valueLength, as the adjustment_factor is overwritten to be 1. We believe the difference in number of characters allocated in the HTML output may not be enough to determine whether the graph is strongly connected."

Evaluation: UMD accidentally switched the responses to Airplan 3 Q 002 and Airplan 4 Q055, this is noted and corrected. UMD correctly concluded that the application does not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.4 Airplan 5

A.4.6.2.4.1 Question 005 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "We believe there is a vulnerability. We assume that the victim user, after uploading a route map, will go through by clicking 'Next' in the following pages, and thus reach the one with path '/passenger_capacity_matrix'. The rows of the matrix generated correspond to the number of airports, and the formatting of those rows is using the method 'format' in FlightMatrixCoach, which calls the method with the same name of the class CellFormatter. The values passed to the latter method call, in particular -1 for len and false for adjust, cause the length of those formatted strings to not depend on random variables. We believe it is thus possible for an attacker to determine the number of airports from observing the size of the HTML output."

Evaluation: There are vulnerable variants of this challenge application where the route map packets from the server are padded to allow unique mapping of packet size to number of airports. In this variant, this is not the case. Without the fixed padding of each cell, we do not believe it is possible to determine the number of airports in a user's route map.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.5 Collab

A.4.6.2.5.1 Question 028 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis, resource bound analysis (analyzed 109 methods in 34 minutes), decompiler and jProfiler. This program uses a BTree to store and query the events. AuditID events are hidden from the user. However, the trees have a maximum size for the inner nodes. This means that if a new event is added into a full node, the node is split and the two new nodes are created. Thus, user Picard has 14 events to start, but one event is hidden, as found in the data in the jar file. A binary search can be used to isolate the hidden event. The internal max size of a node is 9 so initially there are two nodes. Picard can then add events to one of the nodes by starting at the higher or lower bounds. He can see the resulting times it takes to add an event as spliting the node will take longer. He also knows how many nodes it should take to fill up the node. If the time differs and the node was not supposed to be full the hidden event is present. Now continue with the two new nodes to keep narrowing the range that the auditID is located. For Example, SchedulingSandbox.addHelpher adds elemests to an eventidList until the eventidlist[ind] == -2, which means that the list is full. Then it first sorts the array and then calls SchedulingSandBox.split."

Evaluation: The cases where a split does not occur are balanced with random writes to a file. We believe that this sufficiently overlaps the distributions of split vs non-split inserts.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.5.2 Question 035 (AC Time, Intended Not Vulnerable, Answered No)

Response: "See discussion for Question 28 about datastructures. It is the case that the size of the BTree increases with each split, but this information is never sent from the server. In fact, in DataNode.takestep, the datanode for audidID is filtered out by "dres instanceof NormalDataUser." This means that the list sent to the user from search only contains their events and not the hidden events."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.6 Info Trader

A.4.6.2.6.1 Question 008 (AC Time, Intended Not Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis, resource bound analysis (analyzed 128 methods in 37 minutes), decompiler and jProfiler. Multiple links can be added at the same level. The program will preserve previous link if the newly added link points to an uncreated file on the system. The new hyperlink will still be created but not shown on sitemap because the pointed file does not exist on the site yet. Because we want to ensure the file not being created while the link was constructed, this attack is sequence-sensitive. We constructed an exploit that the program will halt over 20 minutes in constructing the sitemap."

Evaluation: The E4 variant of the challenge application was modified to prevent multiple links on the same document and remove this unintended vulnerability. We do not believe this is vulnerable.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.7 Linear Algebra Platform

A.4.6.2.7.1 Question 034 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis (246 tainted loops), resource bound analysis (analyzed 157 methods in 24 minutes), decompiler and jProfiler. The main trigger for the space complexity attack is in com.example.util.MatrixSerializer. In the readMatrixFromCSV function, matrix = new double[numRows][numCols]. When numRows and numCols are large, this malloc call will trigger the space attack. Although there are checks to limit the maxsize to 1000, in com.example.linalg.external.operations.ShortestPathOperation, the matrix can be read without enforcing the maxsize."

Evaluation: UMD identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.7.2 Question 050 (AC Time, Intended Vulnerable, Answered Yes)

Response: "In RPCCLientHandler,it maintains the queue of all tasks using a BlockQueue. In the getTask method, when the two clients are launched together, both of them can satisfy the isEmpty check to take the task. However, if there is one task in the queue, only one client can get

the real task, while the other keeps waiting for the server, and the server keeps waiting for the take() method of BlockingQueue returns, which will never happen. Then this is a dead-lock, and it will timeout. To trigger the vulnerability, the partition number can be made larger than matrix size (only one task will be created). This creates a deadlock by BlockingQueue."

Evaluation: The challenge contains an intended vulnerability in the task distribution for matrix multiplication. UMD identified an unintended vulnerability in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.8 Malware Analyzer

A.4.6.2.8.1 Question 023 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis (55 tainted loops), resource bound analysis (analyzed 49 methods in 7 minutes) decompiler and jProfiler. The file output happens when Logger.write_log_entry(response) is called by Server.class. Under certain exceptions the "response" is not reinitialized and instead concatenated with new messages. This can be triggered by a divide-by-zero exception. The Sample class constructor performs additions, but does not permit negative numbers. However, the method does not check for overflows, enabling us to send a 0 value via an overflow to Analysis.is_packed. Ultimately, this value is sent to Analysis.compute_average_byte_helper to trigger the division by zero. This exception is caught by the log writer in the "add" case. By repeatedly triggering this exception, we can create an output file above the size limit because the error messages are concatenated."

Evaluation: UMD identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.8.2 Question 058 (AC Time, Unintended Vulnerable, Answered No)

Response: "In the space complexity attack described in the Question_023, a divide-by-zero causes the output message to be repeatedly concatenated, exceeding the output size limit. However, the repeated string concatenation do not enable the time limit to be exceeded, and so there is no algorithmic timing exploit with the given input constraints."

Evaluation: Invincea identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.9 PowerBroker 1

A.4.6.2.9.1 Question 019 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis, resource bound analysis (analyzed 3597 methods in 7 hours), decompiler and jProfiler. There is a vulnerability in the method 'tryConnecting' on ConnectionStage. In particular, there is a possibility for an arbitrarily high number of connection attempts, as there is a path through that method (using throw/catch), that recursively calls tryConnecting with the same value of numAttempts. At each attempt of connecting, some information is saved to the log file, and it can thus end up being arbitrarily large."

Evaluation: UMD identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.9.2 Question 040 (SC Time, Intended Vulnerable, Answered Yes)

Response: "We believe there is a vulnerability in ShareData, which is serialized and sent for bid comparison. In the method 'generate', there is a for loop that starts at myOffer and ends at maxOffer, so the time taken by this loop depends on the value of myOffer, and thus information about the latter is leaked."

Evaluation: UMD identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.10 PowerBroker 2

A.4.6.2.10.1 Question 015 (SC Time, Intended Not Vulnerable, Answered No)

Response: "Tools we used: static taint analysis, resource bound analysis (analyzed 3547 methods in 7 hours), decompiler and jProfiler. We could not find a vulnerability."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.10.2 Question 025 (AC Space, Unintended Vulnerable, Answered No)

Response: "We could not find a vulnerability. We focused on the output generated by the logger, and we could not find a place where the file size could be large enough."

Evaluation: Iowa State, Northeastern, and Invincea identified unintended vulnerabilities.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.11 PowerBroker 4

A.4.6.2.11.1 Question 006 (SC Time, Intended Not Vulnerable, Answered No)

Response: "Tools we used: static taint analysis, decompiler and jProfiler. We could not find a vulnerability."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.11.2 Question 013 (SC Time, Intended Vulnerable, Answered No)

Response: "We could not find a vulnerability. The method 'wrapExp' used some randomization in its loops, but the same applies irrespectively of whether the bit is 0 or 1, and the method is only used in generateMasterSecret."

Evaluation: The RSA implementation uses a vulnerable CRT resulting in a sawtooth response time function. The period of this function correlates to one of the two RSA primes. With knowledge of the product of the primes, this can be used to identify the other RSA prime and impersonate the user.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.12 SmartMail

A.4.6.2.12.1 Question 047 (SC Time/Space, Intended Vulnerable, Answered No)

Response: "Tools we used: static taint analysis, resource bound analysis (analyzed 110 methods in 20 minutes), decompiler and jProfiler. Even though the logfile size can reveal whether there is

a mailing list administrator, there does not appear to be a timing channel to reveal the actual email address of the administrator. The list of destination addresses is not sorted when combined with mailing list and individual addresses, preventing an attack based on binary search."

Evaluation: The list of destination addresses is alphabetically sorted and allows for a binary search to identify the administrator's email.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.13 Tour Planner

A.4.6.2.13.1 Question 045 (SC Time, Intended Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis, resource bound analysis (analyzed 458 methods in 33 minutes), jProfiler and decompiler. The execution time of TourCalculator.calcMinSpanningTree is largely determined by root and reqPoints. And iteration through sortedEdge reduces the possible number of edges via ordering. For all combinations of the 5 cities, the running time is widely spread. The running time of the user's request can be compared to the running times of all combinations to reduce the candidate cities."

Evaluation: UMD appears to have identified the side channel that allows mapping the time difference between response times to individual edges. This side channel was not thought to be sufficiently strong; however, Invincea provided an exploit that takes advantage of minute timing differences between the directionality of the edges to satisfy the operational budget. UMD's response did not note the ability to use the timing difference in the directionality of the edges to satisfy the operational budget with an edge-by-edge guessing strategy.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.14 WithMi 1

A.4.6.2.14.1 Question 007 (AC Space, Intended Not Vulnerable, Answered No)

Response: "Tools we used: static taint analysis (410 tainted loops), resource bound analysis (analyzed 1482 methods in 60 minutes), decompiler and jProfiler. We could not find a vulnerability. Functions for file transfer and compression/decompression include tainted loops. However, the file to be transmitted is split into small chunks (256 bytes) before getting compressed, and for this reason the compression ratio to be too low to be able to cause the algorithmic complexity attack with just 10,000 bytes."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.14.2 Question 032 (SC Space, Intended Vulnerable, Answered Yes)

Response: "A message is built representing the state of the current chat/discussion/forum and is sent to everybody, using the tainted method createDiscussionStateMsgBuilder. This message contains a serialization of all IDs of the members of the chat, so the size of this message corresponds to the number of members in the chat."

Evaluation: UMD identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.15 WithMi 2

A.4.6.2.15.1 Question 012 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis, resource bound analysis, decompiler and jProfiler. For the particular set of files in the directory, there is a vulnerability. All 5 files when compressed (method 'pack' of SneakerCompression) and encrypted have different sizes, and the closest of these files are twinkle.txt (1172) and twinkle2.txt (1189) with a difference of 17 bytes. We believe this difference is big enough to be noticed by the attacker."

Evaluation: This version of file transfer first compresses the file and then splits it into fixed size chunks to be sent. Since some files compress more than others, a random amount of padding is added at the end, to prevent an observer from identifying a file by its total compressed size.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.16 WithMi 3

A.4.6.2.16.1 Question 026 (SC Time, Intended Not Vulnerable, Answered No)

Response: "Tools we used: static taint analysis, resource bound analysis, decompiler and jProfiler. We could not exploit any of the possible leakage happening in the math classes and methods."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.16.2 Question 030 (AC Space, Intended Vulnerable, Answered Yes)*

Response: "We could not exploit the reconstitute method of SpiffyUnpacking, but we found a place that might be exploitable. When unzipping the compressed file, first a byte is read to determine the character (C), and the next 4 bytes are then read to determine how many times (N) the latter character appears consecutively at the current location. A while loop is used to write the character C, N times, but if N starts off being negative, the loop will keep decreasing the value of N indefinitely (or until overflow happens), as the check of the loop only checks if N is equal to 0. We could not find a way of interacting with WithMi and setting such an integer value to a negative one. If we were allowed to send our own constructed packages, and not use WithMi's API, we believe a vulnerability is possible."

Evaluation: UMD correctly identified the intended vulnerability. The team's response indicates they were not aware that the attacker can modify their instance of the application as a result the attacker can send their own constructed packages to the application. This response has been change to "Yes" and assigned a post-engagement ruling of correct.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.17 WithMi 4

A.4.6.2.17.1 Question 010 (SC Space, Intended Vulnerable, Answered No)

Response: "Tools we used: static taint analysis, resource bound analysis, decompiler and jProfiler. We assume that the question asks about the particular files in the directory. For this particular set of files and for the given probability of success of 99%, there is no vulnerability. When compressed and encrypted the files twinkle.txt and pi.txt have exactly the same size (4680)."

Evaluation: This version of file transfer splits a file into fixed sized chunks and then compresses them, yielding a unique size fingerprint for each of the included files (in spite of their all having the same original size.) Thus, an attacker observing network packet sizes can determine which file has been sent.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.17.2 Question 031 (SC Time, Intended Vulnerable, Answered Yes)

Response: "We believe there is a vulnerability. In the method addMemberToMemberHistory, if it is a new connection, the user is added to some data structure and then a message is transmitted to them. By measuring the difference in time it takes for the message to be sent, an attacker may associate it with whether the two users have been previously connected."

Evaluation: UMD identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.17.3 Question 036 (AC Space, Intended Vulnerable, Answered No)

Response: "We could not find a vulnerability. There is a check when reading the input characters of a file to be compressed that does not allow for such big files. We could not find anything peculiar with how the files are zipped and unzipped."

Evaluation: UMD did not observe that an integer overflow vulnerability in the RLE compression can be used to trigger a vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.17.4 Question 041 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "Tools we used: static taint analysis, resource bound analysis, decompiler and jProfiler. We believe there is a vulnerability. The method fastModularMultiply contains a for loop with up to 10000 iterations (random number between 0 and 10000, that changes every time the check is performed). This loop might in some cases be executed many times, and as a result leak information about when it is executed. As the method fastModularMultiply is used during the exponentiate call of both encrypt and decrypt. The functionality of the loop is completely unnecessary, as the mod of u with this.M is taken in the end. It seems the probability for this loop iterating at least m times is approximately $\frac{9999!}{(9999-m)!*10000^m}$. For m=10 this is close to 99% probability, which means the attacker can possibly detect the number of bits in the keys being 1."

Evaluation: This variant of the challenge does not implement CRT, and the communications framework is believed to prevent a potential vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.18 WithMi 5

A.4.6.2.18.1 Question 009 (AC Space, Intended Not Vulnerable, Answered No)

Response: "We could not find a vulnerability. We tried exploiting file transfer and compression/decompression, but in this version, the file to be transmitted is split into small chunks (256 bytes) before getting compressed, and for this reason we believe the compression ratio to be too low to be able to cause the algorithmic complexity attack with just 10,000 bytes."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.18.2 Question 014 (AC Space, Unintended Vulnerable, Answered No)

Response: "We could not find a vulnerability. There is a check when reading the input characters of a file to be compressed that does not allow for such big files. We could not find anything peculiar with how the files are zipped and unzipped."

Evaluation: Invincea identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.6.2.18.3 Question 027 (SC Time, Intended Not Vulnerable, Answered No)

Response: "We could not find a vulnerability. Various checks for when two users have been previously connected happen, but the time of the response message, does not greatly depend on them. In particular, when a user has not been previously connected with another, and as a result is added in a data structure, this addition happens after the message is sent."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.6.2.18.4 Question 046 (SC Space, Intended Not Vulnerable, Answered No)

Response: "A message is built representing the state of the current chat/discussion/forum and is sent to everybody, using the method formForumStateMsgBuilder. This message contains a serialization of all IDs of the members of the chat, but if the members of the chat are less than 4, the message is padded with fake IDs. We could not find any vulnerable places."

Evaluation: UMD correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.7 University of Utah

A.4.7.1 Utah Overview

Utah answered 7 questions in E4 with a 71% accuracy. The team focused on AC vulnerabilities (5 of 7 responses) with a 50% accuracy for AC Space an 100% accuracy for AC Time. Utah answered 3 fewer questions in E4 than in E2 but 4 more questions than in E3. The team's accuracy decreased from 80% in E2 to 75% in E3 to 71% in E4; however, they answered 2 SC questions in E4, an increase from E2 and E3 where they answered 0 SC questions.

Table A-75: Engagement 4 Utah Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	1	100	1	100
SC Time	1	0	0	0	0
SC Space/ Time	0	0	NA	0	NA
AC in Space	2	1	50	1	50
AC in Time	3	3	100	3	100
Total	**7**	**5**	**71**	**5**	**71**

Table A-76: Engagement 4 Utah Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	5	4	**80**	4	**80**
Not Vulnerable	2	1	**50**	1	**50**
Yes Answer	5	4	**80**	4	**80**
No Answer	2	1	**50**	1	**50**

A.4.7.2 Utah Specific Responses

A.4.7.2.1 Airplan 2

A.4.7.2.1.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Response: "Our loop tool detected six recursive functions and the tool flags them as potential hot spots. One of the suspects is Orderer.changingRank.changingRank(), which involves a sorting algorithm, which is a frequent source of vulnerabilities. Based on the results from that tool, we manually inspected changingRank(). This allowed us to determine that its worst-case runtime cost is bounded by a recurrence equation of the following form where n is the length of the list being sorted, $O(n)$ is a linear function of n and $T(n)$ is the total runtime of changingRank on a list of length n:

$$T(n) = T(\frac{n}{6}) + 2T(\frac{5n}{6}) + O(n)[when\ n\ mod\ 6 == 0]$$

$$T(n) = T(\frac{n}{6}) + T(\frac{5n}{6}) + O(n)[when\ n\ mod\ 6 != 0]$$

Equation A-1: Recurrence Equation

Though we are still in the process of developing tools to automatically extract and solve this sort of equation, a manual solution shows that it is super-linear in `n` and grows rapidly when the recurrence frequently goes through the first case rather than the second case. So if the length of the list is divisible by 6 and recursively makes the length of the second part […] to be divisible by 6, it makes the sorting algorithm to run for a long time. […]"

Evaluation: Utah identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.7.2.2 Collab

A.4.7.2.2.1 Question 028 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "Our loop visualization tool marks the loop in `fileWrite` as a loop to be checked, and taint flow shows that this loop is controlled by a branch in `addHelper`. We tested to determine whether the delay caused by the loop was sufficiently large to be detectable, and it was. Inspection of the code then showed that the branch in `addHelper` is based on whether a

`DataNode` should be `split`. We developed an algorithm that determines (based on whether a node split occurs or not) which child of a split node an audit node is in. For this algorithm, suppose a node has three low elements, two high elements and an audit element between them. We choose a division point, m, and insert the following sequence of nodes: m-1, m, m+2. This will cause a node split into two children. If the audit element is smaller than m, the split will be between m-1 and m. Otherwise, the split will be between, m and m+2. Next, we insert m+1. If m goes into the left child, the split was between m and m+2, and thus the audit was above m. If m goes into the right child, the split was between m-1 and m, and thus the audit was below m. We determine whether m+1 went into the left or right child by counting how many elements must be added to the left child before it splits. From that we constructed a binary search. However, because it costs 5 operations when an audit is in the left child and 9 operations when an audit is in the right child, our calculations showed that splitting the range evenly would go beyond the available budget more than 10% of the time. Thus we constructed a tool that computes the optimal position for splits. This was usually about a 60%/40% split, but our tool computes exact split positions and deals with quantization effects. We tested this attack against an instrumented version of the code and used that to debug the attack algorithm. In the final version, over the range 0 to `MAXINT`, the attack takes at most 200 insertions. Which is within the operation budget. Since the user `picard` already has some event entries, it is most convenient to run the search over the range 1100004 to `MAXINT`. This still coverers 99.94% of the range, which is within the success budget."

Evaluation: The runtimes for split vs non-split operations are believed to be sufficiently overlapped to prevent a vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.7.2.2.2 Question 049 (SC Space, Intended Not Vulnerable, Answered No)

Response: "This question asks whether there is a side channel in the sizes of packets that could allow the user `picard` to determine the event id of one of his audits. Neither high-level visualization nor code inspection turned up any vulnerabilities, so we attempted to prove the absence of vulnerabilities. The `CollabServer.channelRead0` method takes an `ChannelHandlerContext` object as argument. The `write` method of that `ChannelHandlerContext` object is what writes out the network packets. Tracing the packets that flow to the `write` method, the packet sizes, and what branches or values influence them reveals that (with one exception) the packets are not influenced by any audit values. The one exception is when an error occurs due to inserting a new user event with the same event id as the audit event. However, this is not strong enough to meet the specified operation and success budgets."

Evaluation: Utah correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.7.2.3 Linear Algebra Platform

A.4.7.2.3.1 Question 034 (AC Space, Intended Vulnerable, Answered Yes)

Response: "One of our analysts found the allocation in the function MatrixSerializer.readMatrixFromCSV and noticed that most calls to this function set the condition enforceSize to true, but the ShortestPath operation calls this function with enforceSize set to false. Upon examining how the function uses this condition, it was noted that this allows the user to bypass the size check on the second matrix in a ShortestPath operation."

Evaluation: Utah identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.7.2.4 Malware Analyzer

A.4.7.2.4.1 Question 023 (AC Space, Intended Vulnerable, Answered No)

Response: "The question asks if there is an algorithmic complexity vulnerability in space that would write more than 80kB to a file. The only file written to by the application is the log file response.log. If we follow the taint flow back to see what affects this, we find the log file is written to only in the `Logger.write_log_entry` method. It uses a `FileWriter` object, which overwrites the previous file instead of appending to it, so we can just consider only single `write_log_entry` invocation. Tracing further shows that the `Logger.write_log_entry` method is called only in the `Server` class. In particular there is a big `switch` block containing several such calls. Outside of the `switch` the log messages are simple and short, so we need only consider the calls inside the class. This `switch` block has three categories of branches: *add* and *add_dasm*. These add entries to the database. They only generate only short log messages, so they could be part of the preparation for an attack but are not attacks in themselves. *query* and *query_dasms* These generate only short log messages. *get_function_entrypoints*, *get_cfg*, and *list_dasms*. These cases can generate longer log messages. However, the lengths of their log messages depend on the added `dasm`s (due to the contents or the number of `dasm`s), and the lengths of the log messages are much smaller than the lengths of the `dasm`s added."

Evaluation: The challenge contains an integer overflow vulnerability that can be used to trigger a divide by zero exception. The entire stack trace is then written to file.

Post-Engagement Analysis Ruling: Incorrect

A.4.7.2.5 SmartMail

A.4.7.2.5.1 Question 059 (AC Time, Unintended Vulnerable, Answered Yes)

Response: "From the description, we understood that there were three main operations: sending email, receiving an email, and printing the list of public email addresses for a given mailing list. Each of these operations was clearly separated in our visualization of the code. The layout of the loop graph in our visualizer tool also showed that there was no recursions or cycles in the loop graph so we discarded the idea that the vulnerability could be executed using a single operation. Instead we looked for ways of increasing the cost of a series of operations. A search through the code for the address book showed that there is no way to add new email addresses after it has been initialized. That left us with only the handful of users already provided in the application. Profiling a few example requests showed that most of the time was being spent serializing and deserializing emails, which narrowed our search. We traced the code to determine what was being serialized, and discovered that the email address for each recipient of an email was serialized, along with every message in the corresponding email inbox. We now simply needed to send an email to as many users as possible. Examining the email messages, we noticed that (a) each email stores all of its recipients, and (b) the recipients are stored in a list instead of a set, so they may be duplicated. Also, sending to a mailing list creates an email for every user on the mailing list. So we took the largest mailing list provided and repeated it as many times as possible in the `TO` field without exceeding the limit on the length of a single request. Then we repeated this request as many times as possible within our total input budget."

Evaluation: Utah reported the same unintended vulnerability as Northeastern.

Post-Engagement Analysis Ruling: Correct

A.4.7.2.6 Tweeter

A.4.7.2.6.1 Question 029 (AC Time, Intended Vulnerable, Answered Yes)

Response: "A lightweight fuzzer revealed that character sequences that deviated significantly from dictionary words incurred a substantial response delay. Using a debugger to trace execution, we discovered the call to […] the `com.tweeter.service.Spelling` class, which supplies correction suggestions for each word in a tweet, to contain the delay. Inspection of the decompiled source revealed how correction suggestions were determined. First, character sequences of length greater than two more than the length of the longest dictionary word are disregarded, as are character sequences that appear in the dictionary. If the given character sequence meets neither of those criteria, the `correct` method calls the `isPossible` method which itself uses the `possiblePrefixes` method to determine whether the word is within two edits of a dictionary word. The time vulnerability lies in the fact that it does so by checking `possiblePrefixes` for successively longer prefixes of the given character sequence. This points to an exploitation strategy: choose a long word (which has more prefixes) such that as many prefixes are possible. However, the running time of `possiblePrefixes` is sensitive to the type and number of edits needed to transform a character sequence prefix into a dictionary word prefix. This sensitivity exists because `possiblePrefixes` and its inferior calls short-circuit the generation of edits when a dictionary prefix is found. An inspection of the edit-generating workhorse method `generate1Edits` shows the order in which edits are generated: character removals, adjacent character swaps, character replacements, and finally character additions. […]"

Evaluation: Utah identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8 Vanderbilt University

A.4.8.1 Vanderbilt Overview

Vanderbilt answered the most questions in E4, 52, with a 73% accuracy. Against the segmentation of all questions, Vanderbilt had the highest TNR, 75%, and the second highest TPR, 57%. Vanderbilt's accuracy decreased 10 percentage points from E2, but increased 15 percentage points from E3. Vanderbilt answered 10 more questions in E4 than in E2 and 40 more questions than in E3.

Table A-77: Engagement 4 Vanderbilt Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	11	9	82	9	82
SC Time	13	12	92	12	92
SC Space/ Time	2	2	100	2	100
AC in Space	10	5	50	4	40
AC in Time	16	11	69	11	69
Total	**52**	**39**	**75**	**38**	**73**

Table A-78: Engagement 4 Vanderbilt Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	29	21	**72**	20	**69**
Not Vulnerable	23	18	**78**	18	**78**
Yes Answer	26	21	**81**	20	**77**
No Answer	26	18	**69**	18	**69**

A.4.8.2 Vanderbilt Specific Responses

A.4.8.2.1 Airplan 1

A.4.8.2.1.1 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Response: "We analyzed this application using our profiling tool jBond. This tool attaches to the Java process as an agent and instruments the program to record resource consumption information. Results are output to a JSON file, which can then be studied with scripts or with the Janalyzer. We did several runs of the application with jBond connected. The seemingly most interesting example input files w.r.t. triggering high resource consumptions are many_airports_map.txt and many_flights_map.txt. For both these maps, we profiled loading the map, then running several of the analyses that airplan_1 provides on them. […] It records several "features" of the code. The one that we used here is "method time", which records the total time spent in each method, excluding time spent in callee methods. This is the summation of the times spent in this method for all calls. […]

The top method is Airport.hashCode(). We guessed that this was due to many calls, due to (a) HashMap<Airport> structure(s). […] Some study of the decompiled code shows that the AirportSerializer is a component of the AirDatabase storage, which is used by several other components. Components that use the AirDatabase include the CrewOverseer analysis, which finds an optimal assignment of crew to the collection of flights. Since top 5 methods here are related to this component, we decided to study it further.

We can now focus jBond by only instrumenting the CrewOverseer class. Then we can try some different input structures, to see if we can maximize time consumption. […] An observation from running the different examples is that the number of airports, as well as the number of flights, is limited to 500. The first thing we tried was to build a fully connected graph (all airports have a connection to all other airports), but due to the described limitation, we can only use 22 airports in this case (22*22-22 = 462). This is not enough to exceed the output budget. We then built an input with 500 airports and 500 flights. Given the intention of the CrewOverseer class, we gave this file a structure that promotes re-usability of the crew: 250 flights go from the first half of the airport to airport 250, then 249 flights go from airport 250 to the second half of the airports. The CrewOverseer takes 14m54s to process this file. Far outside the 500s budget."

Evaluation: The challenge program contained a vulnerability in the implementation of shortest path using an A* algorithm vulnerable to a shortest path request with an inconsistent heuristic. Vanderbilt identified an unintended vulnerability in the *CrewOverseer* class associated with crew assignment. Vanderbilt observed that multiple methods that contributed to the runtime for

AirPlan are associated with the CrewOverseer class. Vanderbilt generated an input to increase the calls to these methods by promoting crew re-usability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.2 Airplan 2

A.4.8.2.2.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Response: "For Airplan 2, class com.networkapex.sort.Orderer is used for sorting edges of a vertex in a graph that represents routes between airports. It is performed whenever a user tries to find the capacity of a route from one airport to another (regardless of the weight metric used). When the number of edges of the "source" airport is a multiple of six, the sorting algorithm takes exponential time.
The simplest attack is to create a number of airports and designate one airport, 'a', as the source to all the others. As an example, I created an airport with 198 edges (i.e. a multiple of 6). Uploading this map and subsequently performing the "capacity" calculation with a as the origin, makes the program run for approximately 45 minutes on the NUC. The input file is approximately 5,000 bytes."

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.2.2 Question 038 (SC Space, Intended Vulnerable, Answered Yes)

Response: "Active operations are not useful for this question because access to the system is compartmentalized by user. […] Therefore, we focused on passive operations. After reading the question and trying out the system, we noted that large portions (including many use cases) of the system are irrelevant to the passive operation, simply because they are not guaranteed to occur. […] we decided to focus on the remaining green nodes: upload the map and download the capacity matrix. We wrote a Python script that generates random maps in txt format. […] Profit's aggregate_space_by_phase transformation showed a direct correlation between the secret (number of cities) and the total number of bytes downloaded during the fourth phase (i.e., downloading the matrix). This suggested that the sum of packet sizes of the fourth phase of each interaction is the right observable for this question.
This led us to inspecting the HTML code for the capacity matrix page, which showed that each cell of the matrix is padded to the exact same width of 10 characters between TD tags, using blank characters (spaces), which do not appear in rendered HTML but still count towards the byte size of the HTML code that is sent through the network. This is a strong side channel. The total number of bytes downloaded during the fourth phase (downloading the capacity matrix) can be traced back to the number of airports because there is a nearly-constant number of bytes per cell and the number of cells is quadratic in the secret. This seems to dominate the total byte count of the matrix page, which ends up being directly proportional to the secret. […] We applied Profit's Channel Leakage Quantifier to the profiling data, the results of which are given in the plots below. We see a plot of the raw data along with a fitted model of the relationship between observation and secret. The second plot visualizes the maximum probability of guessing the secret as a function of observation. We see that this probability is always above the threshold probability and thus our tool tells us that the side channel is exploitable."

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.3 Airplan 3

A.4.8.2.3.1 Question 002 (SC Space, Intended Vulnerable, Answered Yes)

Response: "After reading the question and trying out the system, we noted that large portions (including many use cases) of the system are irrelevant to the passive operation, simply because they are not guaranteed to occur. Preliminary exploration also showed that the most interesting use case (and, as we checked later, the only use case that can leak the secret) is the Map Properties page. Therefore, we decided to focus on uploading the map and obtaining its properties. […] We ran 500 interactions with {2, 3, 4, 5, 6} airports, with random edge densities between 0% and 100%, random numbers of digits between 1 and 10, and random airport names between 1 and 3 characters. […] Profit's aggregate_space_by_phase showed a correlation between the secret (strong connectedness) and the total number of bytes downloaded during the second phase (get map properties). […] In the first plot below computed by Profit's Channel Leakage Quantifier, we see the raw samples plotted (a fitted model was not necessary in this case), with 0 representing "Not Connected" and 1 representing "Connected". In the second plot, we see the probability of guessing the secret as a function of observation which is always above the threshold of 99%. Thus, we can conclude the side channel is exploitable within the budget."

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.3.2 Question 016 (SC Space, Intended Not Vulnerable, Answered No)

Response: "After reading the question and trying out the system, we noted that large portions (including many use cases) of the system are irrelevant to the passive operation, simply because they are not guaranteed to occur. […] Preliminary exploration also showed that the gray nodes below are irrelevant as well – they don't contain any information that affects the question. […] Therefore, we decided to focus on the remaining green nodes: upload the map and download the capacity matrix […] Profit's aggregate_space_by_phase showed no correlation between the secret (number of cities) and the total number of bytes downloaded during the fourth phase (i.e., downloading the matrix). […]"

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.4.8.2.3.3 Question 024 (AC Time, Intended Vulnerable, Answered Yes)

Response: Vanderbilt provided the same response and exploit as for Airplan 1 Question 020. Adding that: "The CrewOverseer takes 10m44.728s to process this file. Far outside the 500s budget"

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.4 Airplan 4

A.4.8.2.4.1 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

Response: Vanderbilt provided the same response and exploit as for Airplan 1 Question 020. Adding that: "The CrewOverseer takes 24m56.713s to process this file. Far outside the 500s budget."

Evaluation: The application contains an intended vulnerability in the Dijkstra's implementation of shortest path. Vanderbilt identified an unintended vulnerability in the *CrewOverseer* class associated with crew assignment. Vanderbilt observed that multiple methods that contributed to the runtime for AirPlan are associated with the CrewOverseer class. Vanderbilt generated an input to increase the calls to these methods by promoting crew re-usability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.4.2 Question 055 (SC Space, Intended Not Vulnerable, Answered No)

Response: "After reading the question and trying out the system, we noted that large portions (including many use cases) of the system are irrelevant to the passive operation, simply because they are not guaranteed to occur. […] We hoped that Profit's aggregate_space_by_phase would show a correlation between the secret (strong connectedness) and the total number of bytes downloaded during the third phase (obtaining the map properties). However, this did not happen. […] In the second plot, we see the probability of guessing the secret as a function of observation which is above the threshold of 99% for most observations, but for an observation of 4470 there are a significant number of collisions between the two classes and the probability, p = 0.68207, to guess the secret is far below the success threshold. Thus, we can conclude the side channel is not exploitable within the budget."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.5 Airplan 5

A.4.8.2.5.1 Question 005 (SC Space, Intended Not Vulnerable, Answered No)

Response: "[…] After reading the question and trying out the system, we noted that large portions (including many use cases) of the system are irrelevant to the passive operation, simply because they are not guaranteed to occur. […] Profit's aggregate_space_by_phase revealed a correlation between the secret (number of cities) and the total number of bytes downloaded during the fourth phase (i.e., downloading the matrix). This suggested that the sum of packet sizes of the fourth phase of each interaction is a relevant observable for this question. […] We applied Profit's Channel Leakage Quantifier to the profiling data, the results of which are given in the plots below. We see a plot of the raw data along with a fitted model of the relationship between observation and secret. The second plot visualizes the maximum probability of guessing the secret as a function of observation. We see that this probability is below the threshold probability (99%) for some observations, from which we can conclude that the side channel is not exploitable (in the worst case) to distinguish between two secrets. However, the budget allows for 1 oracle query besides the passive operation. Thus, to show the absence of a vulnerability, we need to show that at least a three-way collision can take place in the worst case. The aforementioned analysis does not suffice to guarantee this. We ran Profit again, 100 times per secret, for the secrets {13, 14, 15}, using random edge densities between 25% and 75%, and

keeping the number of digits of passenger capacities constant (i.e., same number of digits for all cells in the matrix). We did this for number of digits ranging from 1 to 10. This experiment revealed several 3-way collisions when using many digits for small matrix sizes and few digits for bigger sizes. […]"

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.6 BidPal 1

A.4.8.2.6.1 Question 004 (SC Time, Intended Not Vulnerable, Answered No)

Response: "When a user logs on to bidpal_1 (edu.computerapex.origin.StacMain.main), two primes p and q are generated at random and used to construct that user's identity. This means that, even if an attacker knew a user's primes p and q which generate that user's key, the attacker cannot specify to the system what they wish their own p and q to be. This makes it impossible for the attacker to choose their own p and q to impersonate another user. This part of the code can be localized quickly by tracing the entries of random functionalities in the code. Also, there are several other functions constructing the identity of a user, but they are never called. […]"

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.6.2 Question 033 (AC Time, Unintended Vulnerable, Answered Yes)

Response: "To find the vulnerability, we first ran the YourKit Java profiler on the example script. The profiler showed that the program spent a lot of time processing bid comparisons from other users, especially with methods such as BarterDriver.sendShareData . It was noted that every time a bid comparison was made, the program made calls to the custom exponentiate method which returned a java BigInteger. It was also observed that the program could still take commands from a user while still processing their bid comparisons in the background. Our Janalyzer tool showed that many of the methods called during the process of a bid comparison included calls to multi-level loops, 3 of which (2 3-level and 1 2-level) involved the loop within MgProductGenerator.exponentiate. Upon analysis of the decompiled code, it was noted that the bounds of many of the loops were out of user control. However, for more expensive calls such as MgProductGenerator.exponentiate the loop iterated across each bit in BigInteger derived from a randomly generated 256-bit long BigInteger. We determined that the time vulnerability was exploitable by triggering several bid comparisons. Since the max number of contenders in an auction is set to 4, the scenario would require multiple auctions, with the attacker bidding on every auction at the very end. […] They caused the program to run for 1436 seconds (almost 24 minutes) after the last user request."

Evaluation: The challenge did not contain an intended vulnerability. Vanderbilt identified an unintended vulnerability in the bid comparison algorithm. Vanderbilt observed that the decryption algorithm called from *EncryptionPrivateKey.decrypt* makes calls to "an expensive […] custom exponentiate method". This can be used to trigger a vulnerability by making "45 bid comparisons in very quick succession […] and the resulting bid comparisons take over 20 minutes to process."

Post-Engagement Analysis Ruling: Correct

A.4.8.2.7 BidPal 2

A.4.8.2.7.1 Question 044 (AC Time, Intended Vulnerable, Answered Yes)

Response: Vanderbilt provided the same response as BidPal 1 Question 33, adding that: "In addition, while exploring the functionality of the program using our driver, we noticed that theprogram will freeze for approximately 50-60 seconds if the user calls "/listauctions" while the program is trying to make bid comparisons in the background. The exploit is demonstrated in the files located in the assets directory. They cause the program to run for 2457 seconds (40 minutes) after the last user submission."

Evaluation: The application contains an intended vulnerability in the auction checksum. As specific user id triggers the checksum to continually fail. Vanderbilt identified an unintended vulnerability similar to BidPal 1 Question 33 in the bid comparison algorithm. Vanderbilt observed that the decryption algorithm makes calls to "an expensive […] custom exponentiate method". Additionally, Vanderbilt noted that the application continues to accept requests from other peers during the bid comparison process. Vanderbilt used these create an exploit that exceeds the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.7.2 Question 057 (SC Time, Intended Vulnerable, Answered Yes)

Response: "[…] The question asks whether there is a side channel in time which allows an attacker to determine another user's bid to within $30, in the case where said bid was not the winning bid (as the winning bid can be viewed by all after the auction ends). Thus, it suffices to determine how a bid amount and the time it takes to perform a BidComparison are correlated. […] We manually identified the place in the code where the BidComparison data is generated and used JAnalyzer to locate related methods from the control flow graph for further inspection. Most of the core work is done in the method BidComparisonData.generate which is called from AuctionDirector.sendExchangeData during the creation of a BidComparisonData. Specifically, near the end of the generate() method, there is a loop which iterates from 1 to maxProposal (the maximum bid allowed), which contains a suspicious condition depending on what the value of myProposal is […].

Using the program slice, we created a driver that allows us to vary bids and measure the time spent comparing them. We observed that a single comparison is lengthy and so large-scale profiling is not feasible. Consequently, using the driver, we measured the time to compare every bid from 0 to 500 to the maximum allowed bid of 500, a feasible experiment to perform within a reasonable amount of time. We created a prototype tool which we tentatively call Profit which has a channel leakage quantification component to automate analysis. The results of this tool are summarized here. From this profiling data, Profit computed a model of time observable (O) as a probabilistic function of bid x,

$$Model: O = 1932.212 + -1.1792\,x + N(-5.1602e-13, 6.1142) + B(0.003992) * N(247.877, 237.461)$$

where $N(mu, sigma)$ is a normal distribution and $B(p)$ is a Bernoulli distribution. […] Profit computes a maximum gain value (in this case, probability to guess the secret) over possible observations with respect to a gain function $gain(guess, secret) = abs(guess - secret) <= 30$. This maximum gain per observation is shown in the third graph. We see that the maximum gain is above the threshold probability of 0.75 for almost all observations. In fact, the

observations which correspond to probability gain that is less than the threshold happen with sufficiently low probability that they can be ignored for the purpose of this analysis. Hence, the side channel is exploitable."

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.8 Collab

A.4.8.2.8.1 Question 028 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "Collab uses a variant of the BTree data structure to store events related to a user. Each user has a separate BTree - the events for one user are completely isolated from those of other users. The audit events are stored in the same way as normal user events. A user has a limited number of ways that they can interact with the system. These are through the ADD command, SEARCH command, DONE command, UNDO command, and QUIT command. We analyzed each in turn to determine if there was any side channel leakage through timing in which picard could learn any information about the location of his audit event. We began by using Wireshark to perform profiling on the SEARCH command, which returns the public event ids within the user specified ranges. We specified ranges including secret audit events and those that did not. In both cases, the timing was almost identical. The DONE command commits the current state of the sandbox to the master calendar and the UNDO command disregards the current sandbox without committing its contents to the master calendar. We experimented with both and could find no leakage, nor could we with the QUIT command. Finally, we considered whether the ADD command might leak information. To understand how it might, we looked into the structure of the BTree. Each node in the BTree can store up to 8 events, regardless of whether the event is an audit or not. When an event is inserted into a node that already has 8 events, the node is split into two child nodes. In the canonical BTree data structure, when a node is split, one of events gets shifted up into the parent node and each child gets half of the events from the node which was split; e.g., if 8 events is the max for a node, when an insertion is done into that node, each child node would receive 4 events and the 9^{th} would be percolated up. However, in Collab's variant of the BTree, when an insertion is done into a full node, the node is split into two, with the left node getting 4 events and the right node getting 5 events. A new node is created which becomes the parent of these two child nodes, which essentially takes the place of the original node which was split; it is important to note that this new intermediary node does not hold any events, but acts as a connector between the nodes above and the new child nodes.

Since audit events and normal events are treated equally in the internal BTree structure, if the internal structure of the BTree was known, an intelligent user may be able to identify candidate nodes containing an audit event through a controlled sequence of insertions which trigger splits in the nodes. Unfortunately, a user is not able to discern when a split is triggered through normal means. This strategy is only viable if a timing side channel is present which allows a user to determine whether or not an insertion triggers a split. During a Collab interaction, a log file is produced in the challenge_program/logs/ directory. When a user adds an event, a random number (between 86-95) from Random.nextInt of identical statements are appended to the log; this only happens when adding an event does not trigger a split. If adding an event does trigger a split, nothing is appended to the log file. We do know that the random number generator is seeded with the constant 100L, and thus produces the same sequence of random numbers every time. Since the exact number of calls to Random.nextInt is known prior to client interaction, we

can determine the exact number of statements being appended to the log file during an ADD which does not trigger a split. The log file is produced in the server directory and is otherwise not visible to the client - a normal user would not have access to this log file. Because of this, we considered the log file to be out of scope of the attacker and any timing-related leakage due to it not a valid side-channel in time. While we cannot use the log file as a potential side channel, it allows us to know exactly when a split occurs, which we can then relate to timing observations from response traffic for an application request. Without the log file, there is no way to determine when a split occurs. When an ADD command triggers a split, more work is done when updating the internal BTree than during an ADD which does not trigger a split. If we were to compare the timing of packets sent and received during the ADD command between these two cases, then the packets corresponding to the ADD causing a split should take noticeably more time. However, if the time it takes for the server to append a random number of statements to the logfile is approximately the same time it takes the server to split a node, the timings for both ADD commands would be indistinguishable.

In order to determine if there is a timing side channel, we combined Wireshark with the use of the log files and manually added event ids to picard's calendar. Each time an event is added we observed a short sequence of packets being transmitted. We determined from the code that the first packet is sent from the client to the server when specifying the ADD command with the event to be added. The second packet is sent from the server to the client after the ADD command is completed; if the ADD did not trigger a split, this packet is sent after a random number of statements are added to the log file. As the neither the server nor the client does any other work during the sequence, the timing difference between the two packets corresponds exactly to how long the ADD command took to complete. We observed that ADDs which did not split varied in timing from ~1.4ms to ~1.6ms, with the majority taking times around ~1.5ms. ADDs triggering a split consistently gave times greater than ~1.6ms, causing them to stand out. However, we did observe cases where the distinction was not clear; we determined these cases to be related to higher values of the random number being generated for ADDs which don't split. Since we know the exact sequence of random numbers being generated, these timing collisions can be solved: if the next random number was close to the upper bound (around 95), then most likely the ADD did not trigger a split, otherwise we can say with high confidence that a split did in fact occur.

Since we can distinguish from timing differences alone which ADD commands trigger splits and which don't, we can employ binary-search strategy to recover the audit event, based on whether the audit event was in the split node or not. As the maximum number of events per node is hardcoded at 8, a split must occur when the 9th is added. Assume we have just witnessed such a split; ideally, we would like to determine if the audit event is in one of the two resulting nodes. Assume it is in the right node. It must be the case that the first 4 public events are in the left node, and the last 4 are in the right node. If the audit was in the left node instead, then the first 3 public events are in the left node, with the last 5 in the right. By adding nodes to the BTree, we can determine which case it is. Add 4 events that in either case would be added to the left node, then a fifth that would only be added to the left node if the 4th public event was in the left node. If that splits the node, then the 4th public event is on the left side, making the audit on the right. If it doesn't then the 4th event node is actually on the right side, which means the secret is on the left. This could also be done focusing on the right node. Split whichever node the audit was in, and continue the process. Since there are a fixed small number of events in picard's calendar, the audit can be recovered within the budget."

Evaluation: Vanderbilt reported a potential unintended vulnerability due to the random number generator being seeded from a known state (100L). With this knowledge, it is unclear how the user is able to detect whether or not a split occurred. If a split occurs, then the random writes do not occur if it does not, then they do occur. In either case, the observable timing distribution for split vs non-split while not perfectly overlapped are sufficiently overlapped that it is not believed this can be fully resolved.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.8.2 Question 035 (AC Time, Intended Not Vulnerable, Answered No)

Response: "This program has no vulnerability. The loops are well-constrained and there is no provision for the user to enter any recursive data or modify the behavior of the loops other than to increase the time duration of the DONE command in linear fashion using the ADD commands. This method cannot be used to exceed the challenge problem's max resource limit."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.8.3 Question 049 (SC Space, Intended Not Vulnerable, Answered No)

Response: Vanderbilt performed the same analysis as in Question 035, adding that: "If the log file previously discussed was within scope of the attacker, then a space side-channel based on its changing size could inform the attacker of when a split occurs. Using this knowledge, the attacker could perform a binary search like attack to determine the location of the audit event."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.9 Info Trader

A.4.8.2.9.1 Question 008 (AC Time, Intended Not Vulnerable, Answered No)

Response: "Janalyzer showed […] 11 groups in Recursions. […] most of the recursion takes place in post document commands, although DocumentStore.loadDoc is one case where it occurs in get document. There is no recursion in get sitemap command, although it outputs a file that is created by post document which can be essentially unbounded in length. The following results were displayed for Loops […]. All of the above loop entries are all "suspicious" meaning a subcall potentially modifies a loop parameter, so there seems to be significant levels of suspicious looping. […] These loops also seem to be well constrained in that the nested information that is placed in the database is pulled directly from the ged file that is input to it. Therefore, it cannot recurse more than the constraints of the file. I don't think there is any vulnerability here."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.9.2 Question 042 (AC Space, Intended Vulnerable, Answered No)

Response: Vanderbilt performed the same analysis as in question 008, focusing this time on space resource usage rather than time resource usage.

Evaluation: Vanderbilt did not observe that the challenge program contains a broken input guard that can be exploited to trigger a complex document traversal structure.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.10 Linear Algebra Platform

A.4.8.2.10.1 Question 050 (AC Time, Intended Vulnerable, Answered Yes)

Response: "The Linear Algebra program exposes a matrix multiplication operation: It takes two square $n\ x\ n$ matrices as input and returns the multiplication of the two. It creates two "matrix multiplication" servers that are assigned a partition of the work to be done for the matrix multiplication (i.e. it is distributed). The number of multiplication tasks (there are a total of n * n) for a server is determined by:

$$int\ partition\ =\ (int)Math.max(n\ *\ n\ /\ this.numThreads, n\ *\ n\ *Math.exp(S));$$

Equation A-2: Linear Algebra Platform Challenge Partition Equation 1 of 2

Where S is:

final double S = Math.abs(MatrixHeuristics.evalMatrix(matrixA)) - 0.69314718056;

Equation A-3: Linear Algebra Platform Partition Equation 2 of 2

If partition is determined by $n\ *\ n\ /\ this.numThreads$ then the two threads will be assigned an equal amount of work---this would be the case where the multiplication is fastest. However, if $n\ *n\ *\ Math.exp(S)$ determines the partitioning, then the workload will be uneven between the tasks. It turns out that matrixA can be constructed such that all work is assigned to one thread, while the other receives none---this is the worst case scenario for the multiplication.evalMatrix from above […]"

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.11 Malware Analyzer

A.4.8.2.11.1 Question 023 (AC Space, Intended Vulnerable, Answered Yes)

Response: Vanderbilt investigated several potentially vulnerable sections of the application and concluded that they were not vulnerable. They observed that: "The Server class writes the contents of the "response" field to a log file. For most requests, the response is appended by a short fixed-length string. For an "add" request, however, there is a catch block that appends the stack trace of an Exception to the response. We have analyzed this case by putting the code inside the try block in a driver (while catching the same 2 specific exceptions as the original) and running SPF on it. This shows that there is another type of exception that may be thrown: an ArithmeticException caused by division by zero. When these exceptions are thrown a few times in a row, the output budget can be exceeded."

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.11.2 Question 058 (AC Time, Unintended Vulnerable, Answered No)

Response: "[…] From Janalyzer results, there are only 2 basic flows indicated as type1 and type2. The type2 loop is just used to extract the 256 Sample value occurrances saved in an array and output them as a comma-seperated ASCII list. This loop is well-behaved and is not a source of vulnerability. The input budget only allows for 6 add/query requests.

With jBond and the example scripts, the methods that consume most time are those in the Analysis class: compute_cosine_similarity_helper and compute_byte_entropy_helper. We cannot, however, control their complexity; it is bounded by constants (255 and 664). Another method in which a lot of time is spent (and which calls compute_cosine_similarity_helper several times) is Database.compute_all_similarities . For a certain Sample, it computes similarity to all other Samples. It stores results in a TreeSet (red-black-tree). So, worst case would be to make the tree imbalanced all the time. Ordering is on this double score. Tried with max test input (5 samples, 1 query), but no vulnerability. Server has a custom MD5 checksum computation. It looks suspicious, but has no vulnerability. We manually checked every nested loop and recursion pointed by janalyzer. The most promising lead is in its CFG construction part.

We use two tricks:

(1) let every instruction cause a new CFG to construct.

(2) use a jump at the end to make each CFG include all the instructions

However, within the budget, only 6s are observed, so we think there is no timing complexity vulnerability"

Evaluation: Invincea identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.12 PowerBroker 1

A.4.8.2.12.1 Question 019 (AC Space, Intended Vulnerable, Answered No)

Response: Vanderbilt used JAnalyzer to identify the recursive methods and loops within the application. "In order to find the space vulnerability, we need to be able to cause the output to exceed 1000000 bytes without going over the 800000 bytes of packet information sent by the attacker. To look for a weakness we need to look at the data that is output to the file to see if there are one or more entries that can have very large data contained in them. These are text files that are output, so anything that outputs short fixed length strings can only accomplish this if the data is issued a very large number of times. So, the things to look for are:

1. potentially long iterations of the output commands where the loop size is controllable by the user.

2. output of a string that is specified by the user that can be made arbitrarily long

Since the input size is slightly smaller than the desired output size, the output would either have to have a repetition of the input or the input would need to be compressed during transmission so that the output size exceeds the compressed size by approximately 1.5 or more, depending on the amount of overhead in the packet transmission. Looking for what type of data is output to the files that signify the basis of the vulnerability, I did a search on println and pullOutputStream which is the getter function for the output file saved in org.digitalapex.head.Control. Any data output to this could implicate a vulnerability if the data can be made very long of recursive, but

note that this data must be produced in one of the non-attacker's output files, so it must be data from the attacker or induced by the attacker. This eliminates many of the recursive entries of executing a script command, since those only redirect the data into the attacker's output file."

Vanderbilt ran further analysis on the bidding process and concluded that the application was not vulnerable.

Evaluation: Vanderbilt did not observe that if the attacker connects to the victim user and disconnects before the connection establishment is complete, the victim application will continuously attempt to re-connect, failing each time and logging the failed attempt. This log file is written to disk.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.12.2 Question 040 (SC Time, Intended Vulnerable, Answered Yes)

Response: "[…] The question asks whether there is a side channel in time which allows an attacker to determine another user's bid to within $30, in the case where said bid was not the winning bid (as the winner and winning bid amount is logged in a logfile for each user). Thus, it suffices to determine how a bid amount and the time it takes to perform a BidComparison are correlated. For any given user who makes an offer during some auction, they are guaranteed at least 2 BidComparisons with every other user: one BidComparison received after initially sending a BidCommitment, and a BidComparison when another user submits a BidCommitment. When an auction is first started, the seller places a "reserve" bid which is logged in each user's logfile. Thus, an attacker not only knows the value of his own bid but also that of the seller's bid. We manually identified the place in the code where the BidComparison data was generated: the generate method in the class ShareData is where the bulk of the work is done. Specifically, near the end of the method, there is a loop which varies in number of iterations depending on what the user's bid is. […] The method differ.getDelta from the Deltafier class essentially performs repeated multiplication, which is done ~ maxOffer/4 times. Based purely on speculation, the amount of time taken in each loop iteration is a function of the first parameter, which in this case is the BigInteger array y. However, the values within the array seem to be nearly all the same magnitude (with small differences between them), and thus the timing should vary little between iterations. Since the amount of work done in the generate() method is proportional to the bid submitted, the calculation of BidComparisons presents a possible timing channel. […] Using the program slice, we created a driver that allows us to vary bids and measure the time spent comparing them. We observed that a single comparison is lengthy and so large-scale profiling is not feasible. Consequently, using the driver, we measured the time to compare every bid from 0 to 500 to the maximum allowed bid of 500, a feasible experiment to perform within a reasonable amount of time. […] Profit computed a model of time observable (O) as a probabilistic function of bid x,

$$Model: O = 1930.437 + -1.1783\,x + N(4.7335e-13, 6.3179) + B(0.005988) * N(206.821, 136.839)$$

Equation A-4: PowerBroker Challenge Model Equation

where $N(mu, sigma)$ is a normal distribution and $B(p)$ is a Bernoulli distribution." This analysis is similar to that performed for a similar question in BidPal.

Evaluation: Vanderbilt correctly the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.13 PowerBroker 2

A.4.8.2.13.1 Question 015 (SC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt performed the same analysis as for PowerBroker 1 question 040 and concluded that the application was not vulnerable.

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.13.2 Question 025 (AC Space, Unintended Vulnerable, Answered No)

Response: Vanderbilt used JAnalyzer to identify the recursive methods and suspicious loops. Vanderbilt investigated the recursive structures and 11 through 8-level looping structures and concluded that these were not vulnerable. Vanderbilt identified points in the program that create output files by searching for *println* and *pullOutputStream*. Vanderbilt also investigated the id values that are user-controlled and the bidding strategy. The team concluded that: "There is an exploit that allows an attacker to create an ID value of his choosing, which will be output by the other instance that is hosting the bidding. There is no limit on the size of this ID value, which can cause the output to exceed any arbitrary limit. However, that ID value needs to be transmitted by the attacker to the other instances and there is no compression on this value. Because the attacker's budget is smaller than the size of the required vulnerability and the ID value is only output once for the transmitted bids, this cannot produce the vulnerability."

Evaluation: Iowa State, Northeastern, and Invincea identified unintended vulnerabilities.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.14 PowerBroker 3

A.4.8.2.14.1 Question 001 (SC Time/Space, Intended Not Vulnerable, Answered No)

Response: "Every buyer or seller who wishes to participate in a Powerbroker run must input a power configuration profile prior to any auctions or even connecting to other peers. A power configuration profile consists of power users, power generators, and budget. Power users require a specific amount of power, while generators are able to provide power, but for a cost. Thus, the cost to generate is defined as the total cost to buy enough power from generators to satisfy the power user's needs. A seller is able to produce enough power from their own generators to give power to their users, and can sell off any extra power their generator has to potential buyers. Buyers aren't able to produce enough power from their generators on their own and must buy power from sellers in order to satisfy their power generation demand. Both power generation cost and demand is computed in in the two functions broker.SimpleProductEvaluator.makeGenerationPlan and broker.SimpleProductEvaluator.generatePromisePlan . This is also the place where sellers compute what auctions they can make with their excess power, and the minimum bid for each of these auctions. While the minimum bid for each auction does depend on the cost for each excess generator to produce power, it has no correlation to the seller's cost to generate. This

computation is done before being connected to any other peers; i.e., it is done offline. Once the cost to generate has been computed, it is not conveyed in any way to the rest of a Powerbroker run. This means that once auctions have started, all information regarding a seller's cost to generate is not reachable, and since the number of auctions a seller has or the price of these auctions is not correlated in any way to the seller's cost to generate, there is no way for an attacker to deduce any useful information from an auction. It should also be noted that the cost to generate does not influence the timing or size of packets sent during an auction. An attacker is also not able to observe the time it takes for the cost to generate to be computed, as that is done offline."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.14.2 Question 003 (SC Time/Space, Intended Not Vulnerable, Answered No)

Response: "[…] Both power generation cost and demand are computed in the two functions broker.SimpleProductEvaluator.makeGenerationPlan and broker.SimpleProductEvaluator.generatePromisePlan . Here, sellers compute what auctions they can make with their excess power, and the minimum bid for each of these auctions; buyers compute their power generation demand here as well. It is important to note that sellers must have zero power demand, while buyers must have a nonzero power demand. Consequently, buyers will never start new auctions, while sellers will never bid on auctions other than their own.

A typical interaction for a single PowerBroker run consists of multiple auctions, with each auction consisting of a single round of bidding. Once an auction has been started, the seller places their minimum bid (in order to secure the auction in the case which there are no potential buyers) while every potential buyer calculates their bid and places it accordingly. The calculation is performed automatically without any user input, located in the function broker.BidPlan.calcAmountToPromise. […] Crucially, this function is the only one which is related to the value of the secret, namely the power generation demand of another buyer (the prior step of calculating demand is done offline before connecting to peers). […] In cases, when no other conditions in the calcAmountToPromise will be satisfied it will return the value of remaining budget, we can get the value of power generation demand by doing reverse calculation of the second last auction values […]. While there is a timing side channel which may allow an attacker to determine another buyer's bid, this is not enough information needed to reconstruct the power demand for the given buyer. We do not believe there is a side channel in space or time which would allow us to determine the power generation demand of another buyer."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.15 PowerBroker 4

A.4.8.2.15.1 Question 006 (SC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt performed the same analysis as in PowerBroker 1 question 040 and concluded that the challenge program was not vulnerable.

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.15.2 Question 013 (SC Time, Intended Vulnerable, Answered Yes)

Response: "In the powerbroker application, a user does not have a password linked to their identity but instead has an id file. This id file specifies two primes which are used to generate a private key (com.virtualpoint.numerical.composeKeyFromPlugin) used to sign messages (com.virtualpoint.numerical.CryptoUtil.sign). The server verifies that this signature is correct when receiving a message (com.virtualpoint.talkers.internal.Dialogs.CryptoState and com.virtualpoint.numerical.CryptoUtil.verifySig). In order to impersonate a user, it is enough to create a matching id file. To do this, it is necessary to obtain the values for the primes p and q. The function com.virtualpoint.numerical.CipherPrivateKey.decrypt signs a message using a private decryption key, d, generated from p and q. (decryption and signing are both done with the private key).

If g is the input, then signing is done by calculating $g^d \ mod \ n$ where $n \ = \ pq$. An application of the Chinese Remainder Theorem allows us to compute $g^d \ mod \ n$ from $g^{d \ mod \ p-1} \ mod \ p$ and $g^{d \ mod \ q \ -1} \ mod \ q$. com.virtualpoint.numerical.CipherPrivateKey.decrypt uses this observation to accelerate the modular exponentiation process involved in signing. Thus both $g^{d \ mod \ p \ -1} \ mod \ p$ and $g^{d \ mod \ q \ -1} \ mod \ q$ are calculated separately. The function used for modular exponentiation is com.virtualpoint.numerical.MontgomeryMultiplier.exponentiate. This function converts g and the current result to their Montgomery forms in order to accelerate exponentiation. Analyzing the code, we see that an addition reduction is done in com.virtualpoint.numerical.MontgomeryMultiplier.montgomeryMultiply depending on whether the output is greater than the modulus. This was suspicious to us, so we investigated further. It turns out that the probability of an extra reduction being performed is proportional to how close g is to the modulus (which recall is either p or q). Thus, as g approaches either p or q from below, the number of extra reductions performed increases and hence so does the execution time. At exact multiples of p or q the runtime greatly falls. This timing information allows us to determine how close g is to one of the prime factors and we can use this information to determine p and q. We have not found any noise strong enough to obscure this side channel, making this attack feasible."

Evaluation: Vanderbilt correctly determined the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.16 RSA Commander

A.4.8.2.16.1 Question 039 (AC Time, Intended Vulnerable, Answered No)

Response: Vanderbilt used JAnalyzer to flag recursive methods and potentially vulnerable loops. They investigate the DES encryption and concluded that it was not vulnerable. They identified a potential vulnerability in the OpenSSLRSAPEM.DER.DERTLV.retrieve method and concluded that it was not vulnerable because the attacker would have to modify the user's private key, a non-allowed action.

Evaluation: Vanderbilt missed the intended vulnerability, that the application has both fast and slow modes for RSA encryption. The attacker can force the slow mode of encryption to be triggered and control the runtime of this slow mode.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.16.2 Question 053 (SC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt performed manual analysis for this question, looking at the entry points, using taint analysis to "identify relevant fragments", analyzing tainted modules. Vanderbilt identified the requestPacketParser.serialize() and requestPacketParser.parse() methods and the ones that modify the secret. Vanderbilt manually analyzed these methods and noted that the BigInteger.pow() method leaks the hamming weight of the exponent, but that it is only used when the exponent is a public key. They added that: "We do not believe there is a side channel in time. We find it worth mentioning though, that there is a way to extract a plaintext within the budget by only observing size packages and the content of the packages."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.17 SmartMail

A.4.8.2.17.1 Question 043 (AC Space, Intended Not Vulnerable, Answered No)

Response: Vanderbilt flagged and investigated recursive and suspicious loop methods and concluded that: "There does not seem to be a space vulnerability in this program. There were no recursive methods and many of the loops had restrictions to prevent the loop count from becoming very large. There also were no methods that seemed to consume an excessive amount of memory."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.17.2 Question 059 (AC Time, Unintended Vulnerable, Answered No)

Response: Vanderbilt flagged and investigated recursive and suspicious loop methods and concluded that: "There does not seem to be any algorithmic complexity in time vulnerability in this program. There were no recursive functions used in the program and the loops that were found were quite constrained. Although the EmailDirHandler class is setup to handle a list of emails, the main doMail method insures that there is only 1 .mime file to be processed. There are only 2 mailing lists with a maximum of 6 entries in the list and new email lists cannot be made. There are also only 7 email addresses defined, and new email addresses cannot be made either. The sender and receiver email addresses must be name@smartmail.com where "name" is limited to a maximum of 25 characters. The receiver must be one or more of the listed email or mailing list addresses, the sender is supposed to be a single address, but can be anything because it is not verified. Although this can allow you to have up to 2 senders by keeping the names short such that the length of both of them including the "smartmail.com" and the ";" separator character is less than 40 characters, it doesn't matter because if the names aren't in the list of 7 names then reading the pending emails simply displays "null" for the sender. Also, the subject text length is limited to 125 characters and the body is limited to 250 characters. There are just too many constraints placed on the input."

Evaluation: Northeastern and Utah identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.18 Tour Planner

A.4.8.2.18.1 Question 011 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "We wrote a Profit script that repeatedly picks 5 cities at random and executes the query, using a generalization of the provided curl example. The script also adds marker packets to delimit the interactions. Since a query includes the full name of all five cities within its URL, and the rest of the URL exhibits a regular pattern, we also stored the length of the query URL as a potential intermediate secret which might provide some insight into the actual secret.

Using Profit and active operations (which have no cost), we are able to request the list of cities and run the aforementioned analysis on them. There is no user-triggerable way to modify the set of available cities (except by changing the contents of the data files on the server), so we will assume that it will not change. '&point=' strings, and the relationship between a space in a city name and the "%20" that appears in the URL in such cases due to URL-encoding. Therefore, we can infer the sum of the lengths of all five city names from the total bytes uploaded during the query. […] The results returned by ABC (that is, the number of possible combinations for each possible sum from 20 to 65) was the following: {1, 0, 5, 5, 15, 25, 45, 75, 120, 185, 271, 395, 540, 735, 960, 1226, 1525, 1850, 2190, 2530, 2861, 3150, 3400, 3580, 3685, 3706, 3640, 3490, 3265, 2980, 2646, 2290, 1925, 1570, 1240, 946, 695, 490, 330, 210, 126, 70, 35, 15, 5, 1}. For several sums, the number of combinations that could have produced that sum is greater than the 2999 oracle queries allowed. For instance, if the sum is 42, the number of possible combinations is 3400, which exceeds 2999. Interestingly, if we don't consider urlencoding, then the model counts (for each possible sum from 20 to 55, which is the maximum in that case) would be the following: {1, 0, 5, 5, 15, 25, 45, 75, 115, 180, 251, 355, 470, 605, 750, 896, 1040, 1155, 1250, 1300, 1306, 1265, 1180, 1060, 910, 751, 590, 440, 310, 205, 126, 70, 35, 15, 5, 1}. Since the worst case is 1306, and we have 2999 oracle queries available, without urlencoding this would be a side channel well within the budget.

At this point we noted that the oracle queries are about the unordered version of the list. And let us recall that the secret that we want to extract is also the unordered version of the cities that the user would like to visit. This means that we can make do with a much smaller number of oracle queries, since we only need one query per unordered combination of city choices, i.e., one query per possible multiset of 5 cities. We asked ABC to enumerate all solutions to each possible sum (from 20 to 65), and in each set of solutions, we replaced each solution with its unordered counterpart. This yielded the following counts: {1, 0, 1, 1, 2, 2, 4, 4, 7, 8, 11, 13, 18, 20, 26, 30, 36, 40, 47, 50, 57, 60, 64, 66, 69, 68, 68, 66, 63, 59, 55, 49, 44, 38, 32, 27, 22, 17, 13, 10, 7, 5, 3, 2, 1, 1}. Thus, in the worst case, which happens when the sum obtained from the side channel is 44, we need 69 oracle queries to be certain that we can disambiguate between the 69 possible (unordered) combinations that add up to 44. Since active operations have no cost, and we only need 1 passive operation to observe the user's query, we can use up to 2999 oracle queries. But we only need 69 in the worst case. Therefore, starting with the sum obtained from the side channel in space, and with the help of the oracle, we can always find out exactly which five cities the user would like to visit."

Evaluation: The unordered set of user request result in 53,130 possible combinations. The model counts provided yielding a worst case of 69 collisions contains only 1287 items. Additionally, the application uses an AES256 cypher with a block size of 128 bits. The worst-case collision for all possible secrets is an observable encrypted output size of 405 bytes. When coupled with the encrypted request sizes, the worst-case pair of encrypted request and respond

sizes was 218 bytes and 405 bytes, resulting in 24,696 collisions. This is not resolvable within 2999 oracle queries.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.18.2 Question 045 (SC Time, Intended Vulnerable, Answered Yes)

Response: "[…] Every query produces a certain pattern of delays between progress report packets (which are streamed by the server to the client). By grepping "progress" in the codebase, we concluded (as per graphhopper.tour.TourCalculator.java) that this is related to the time that it takes to complete each step of the calcMinSpanningTree algorithm. A pattern can be characterized as 5 deltas, i.e., the sizes of the gaps that appear right after each of 5 particular packets of the same size located towards the end of each interaction. We will call a vector of 5 deltas a signature. […] After a few profilings we realized that the last of the 5 deltas does not change: it is always almost exactly the same size. This may be due to the fact that the last step of the tour-planning algorithm (see graphhopper.tour.TourCalculator.java) takes a constant amount of time. In any case, for all practical purposes, there are only 4 deltas available to encode the query into a signature. The crucial experiment would be: Profile all 25^5 possible queries. Save a mapping between queries and signatures. Observe the benign user's network packets during his/her query. Extract the signature from the interaction. Search the mapping to determine which query caused it. The first problem are collisions: multiple different queries could produce the same signature. In that case we must use the oracle operation to disambiguate between the multiple candidates. This will work if and only if no signature represents more than 299 distinct queries (the max number of oracle operations within budget). The second problem is noise. To mitigate it, we would need to sample each query multiple times, perhaps filtering out any outliers, and take the average of the observations (which would become the "representative" of that class of noisy 5-dimensional observations). This means that we will need some notion of distance between points (e.g., a vector norm) and we will have to pick the point closest to the observation, which may or may not lead to the right answer; if it doesn't, we will need to try another close neighbor, and so on (within the budget of 299 oracle operations). If this method predicts the right answer 95% of the time, the answer is YES. Otherwise, the answer is NO. Since the aforementioned requires a few thousand CPU hours, we tried to come up with a smaller experiment through which we can estimate the feasibility of storing such a large number of signatures without causing an exceedingly large number of collisions.

Note that said feasibility is directly tied to the distances in the graph: for instance, if we ran Tour Planner on a completely symmetrical graph, we would probably have an enormous number of collisions. (But the Tour Planner graph is fixed, and comes from real-world data.) We used Profit to run 1000 random queries, sampling each query 10 times, and take the average over those samples as the representative delta vector for that query. We then considered the hyper-cuboids or "balls" of samples around each averaged representative, and the average "radius" of such balls (distance from observations to center value). We compared this with the average distance between centers of different balls. Our results indicate that the average radius was 0.00056202, while the average distance between balls was 0.01133696. This is good news, although still not sufficient to conclude that intersections in the larger experiment would be rare enough. Our experimental results indicate that the total interval of time occupied by the 4 nontrivial deltas is typically between 3 msec and 4 msec. What is the "resolution" of this time in terms of easily distinguishable signatures? How many different values can it represent without causing too much confusion? If all four deltas were roughly the same duration (around 1 msec each), and each of

those milliseconds consisted of about 20 slots, we would be able to represent $20^4 = 160{,}000$ different things.

Assuming our timing information partitions the space into 160,000 buckets, we want to determine if we are able this partition is refined enough for an attacker to guess the value of the secret in their remaining 299 guesses. To do this, we decided to try and compute the probability of a 299-way collision given that there are 25^5 possible queries. This can be phrased as a variant of the birthday problem which computes the probability that two people out of N share a birthday. Here we want to determine if 299 secret values out of 25^5 share a representation. This is hard to compute exactly, so we approximated the probability of collision using a Poisson approximation. Here we mentally consider each set of 299 secrets and call that set a "success" if they all share the same representation. The probability of getting at least one success, i.e. that there is at least one 299-way collision, is about $1 - e^{-\frac{(25^5 choose\ 299)}{(160{,}000)^{298}}}$ which is extremely small. We are assuming that each partition is equally likely, which is probably not the case in reality (this depends heavily on the graph and its distances). Nevertheless, we think it demonstrates that an attack based on a side channel in time might be feasible. Although any particular dimension can contain many collisions, for two different queries to overlap in ALL dimensions is unlikely (typically, at least one of the dimensions can be used to disambiguate). Even when this doesn't happen and all dimensions collide, that still needs to happen to 300 objects at once for it to become a problem."

Evaluation: Vanderbilt correctly identified that there is a timing signature for each request for 5 cities composed of the timings of the intermediate packets received from the server. The team did not observe that there was an additional constraint that resulted in redundant timing signatures for 5-city request containing the same cities regardless of the order. This creates a unique signature for each unordered set and the signature can be sufficiently modeled with a 4-dimensional vector of the mean response time for each edge or an n-dimensional gaussian model for each unordered set of cities (see whitepaper on Tour Planner for further information). Vanderbilt accurately observed the permutation constraint but missed the combinatorial constraint; however, the team accurately observed the existence of a timing signature for the user request.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.18.3 Question 056 (AC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt investigated the recursive methods and suspicious loops flagged by JAnalyzer. They used jBond to profile the runtime for a tour request and note that the "expensive calculations are done on start-up of the server". They also used a version of their TimingChannelListener to generate the set of all possible lists of cities to tour and collect the best and worst-case bytecode instructions. They added that "Best and worst cases are given in terms of bytecode instructions, measured only inside the com.graphhopper.tour.TourCalculator.calcTour(), i.e. not including initialization. The request above, for a tour of all cities with starting point Boston, takes on average 0.14s to run. Likely, a large section of the listed 0.14s is overhead (network, parsing requests, building output etc), not the actual calculation of the tour. In order to exceed the budget, the worst-case would have to be at least a factor 214 worse than the best case; in reality, more because overhead is not taken into account. The differences seen are not large enough, so we can conclude there is no vulnerability"

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.19 Tweeter

A.4.8.2.19.1 Question 021 (AC Space, Intended Vulnerable, Answered Yes)

Response: "In our search for a vulnerability, we first ran the YourKit profiler while exploring Tweeter's user interactions. Of note was com.tweeter.web.TweetingController.twitterRegisterCreate , which initialized and stored an avatar upon every user registration. Janalyzer's News shows one four-level loop and two three-level loops, all three of which deal with Avatar initialization, Tree initialization, and the createImage method in vash.Vash.createImage returns Output.dataToImage, which performs two allocations whose parameters are either byte arrays or width/height dimensions:

DataBufferByte data = new DataBufferByte(pix, pix.length);

...

BufferedImage bimage = new BufferedImage(w, h, 5);

Further profiling on account registrations confirms these initial suspicions, showing many allocations in methods that take the same type of byte array or width/height parameters. Our Janalyzer tool also reports a particularly suspicious recursion in vash.operation.OperationNode.compute , an abstract method used to randomly generate the various shapes found in users' avatars. Further examination in Janalyzer's Loops yields a four-level nested loop, starting at com.tweeter.service.AvatarService.afterPropertiesSet, continuing through the Avatar's constructor, its associated Tree structure's generateCurrentFrame method, and finishing in the compute method of vash.operation.Spiral. Brief manual inspection of some of these operations shows that each computation iterates over the avatar's width and height. After following this call tree, we discovered a conditional within the Avatar class's constructor that implies possible sizes of 128, 256, and 512 […]"

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.19.2 Question 029 (AC Time, Intended Vulnerable, Answered Yes)

Response: "The first thing we noticed about this question is the extremely low budget. We realized that we would only be able to use one operation, so we had to determine which operation had the longest worst-case time. We began by profiling tweeter in order to determine what the most frequently called and costly functions were for each operation. We immediately noticed that the spell check methods, especially those called by com.tweeter.service.Spelling.isPossible , consume most of the program's total CPU use and are called when the user sent a tweet. In addition, our Janalyzer tool reported a 2 level loop in com.tweeter.utility.TrieNode.lookup , one of the more expensive methods called by isPossible. After examining these methods more closely, we realized that while isPossible is only used to determine if it is possible to come up with spelling suggestions for a misspelled word, the way it does so involves a lot of nested loops containing expensive operations, such as calls to com.tweeter.utility.TrieNode.lookup as well as creating and populating new HashSets. We then began to look over the key parts of the code and test tweets in order to figure out if the contents of the tweet could influence the amount of time used by isPossible and its called methods. We

determined that the cost of the spell check loops depends heavily on the input they are given, since isPossible iterates over substrings from the beginning to each subsequent character for each word in the tweet that is misspelled. We also determined the max number of characters that could be fit in a tweet (140) and the max possible length of a word that could be spell checked (26). Once we had that information we were able to start working our way toward a tweet that forced isPossible to run as long as possible as many times as possible. Because the complexity of the spelling functions and their dependence on the contents of the dictionary.txt file made them time prohibitive to walk through step-by-step, we used examples and the profiler to figure out how to format an ideal tweet and what to put into it. Based on the results of our testing, we learned that repeated words are treated exactly the same each time by the spelling methods, which allowed us to use a single word that is a pathological case for the spell check methods repeated until we reached the 140-character limit. Additionally, we learned that the length of that string should be either 14 or 15 characters long because that is the point at which the cost of making the string longer balances with the cost of having more copies of the string. Finally, we learned that certain words and phrases took much more time to analyze than others. We then began to break apart and recombine these phrases with the goal of creating a word that could tie up the spell check methods for the entire runtime limit. After some experimentation with that, we eventually came up with a test case that takes over 25 minutes."

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.20 WithMi 1

A.4.8.2.20.1 Question 007 (AC Space, Intended Not Vulnerable, Answered Yes)

Response: "[…] After finding a vulnerability in the compression component of withmi_3, we decided to study the one in other versions as well. This version is different from those in WithMi 3, 4 and 5. It builds a prefix tree / trie of common characters. Each character is then encoded as a series of steps to the left or right in the trie. The trie is read by the method SpiffyWrapper.readFormation(). There is a bound check in the method that makes sure you read to a depth of at most 6000. However, depth of the trie 6000 means up to 2^6000 - 1 nodes! For each node, a Node object is allocated. We can analyze memory consumption of readFormation() with SPF-WCA. […] It shows that the worst-case memory consumption is 572 + 192x bytes, where x is the number of (symbolic) bytes in the input. The path conditions for the worst-case take the shape ((b0 & 255) & 1) != 1 && (((b0 & 255) >> 1) & 1) != 1 && (((b0 & 255) >> 2) & 1) != 1 etc, indicating that the worst case is when all bits are 0 (SPF-WCA writes the path conditions for different input sizes to Driver.csv in the specified results directory). All bits are zeros corresponds to adding more nodes to the trie, as a 0 bit indicates a node is not a leaf. An exploit was made where a long string of zero bits is written instead of an actual trie. It was made by replacing the SpiffyWrapper class in the JAR. […] Running this JAR, the attacker will send a file to another user. This does not actually send the file, but rather sends corrupted input in which the trie has been replaced by a long string of zeros. As it turns out, as the method is also recursive, the other user's program crashes due to stack overflow. As crashing due to stack overflow is arguably worse than high heap space consumption, we have answered yes to this question."

Evaluation: The challenge did not contain an intended AC Space vulnerability. Vanderbilt identified a vulnerability in the trie structure that allows for up to "$2^{6000} - 1$" nodes. Each node

object resulting in memory consumption. This vulnerability results in a stack overflow and represents a clear bug in the challenge application. While this is noted, this response is given a post-engagement analysis ruling of incorrect due to the fact that the observed memory resource usage is approximately 84 MB, less than the required resource usage of 2000 MB.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.20.2 Question 017 (AC Time, Intended Vulnerable, Answered No)

Response: Vanderbilt investigated the recursive and suspicious loop structures in the application and concluded that: “The recursions and loops found by Janalyzer for this program do not seem to have any vulnerabilities”

Evaluation: Vanderbilt did not observe that the Huffman compression implementation in this challenge can be used to cause an AC Time vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.20.3 Question 032 (SC Space, Intended Vulnerable, Answered Yes)

Response: […] we note that there are 3 different types of actions that can change the number of users involved in a chat -- when a chat is created (0 to 1 user), when a new user is added to a chat and when a user disconnects from the application. We consider the "number of users involved in a chat" to be the sum of the user who started the chat and all unique users added to the chat. […] Profit's space_partitioned transformation revealed that, when a new user was added to the chat, a message informing other members of the chat of the new addition would be sent out. For example, if there were three users in a chat and a fourth was added by one of them, then the other two members (as well as the inviter) would receive messages saying that the fourth was added. Therefore there is a relation between the number of packets sent out when a new user is added and the number of users in the chat. We noticed that if the same user was invited twice to the chat, then there would not be an extra message sent twice for that user. Once we looked into the code, we realized this was due to a dictionary being used to store the relationship between users and the chats they are a part of. We looked into the source code to determine the cause of this behavior and we discovered that when a message is sent to a chat, the method edu.networkcusp.chatbox.HangIn.deliverMessage() is called. This method actually iterates through all members of the chat and sends a message to each. This explains the behavior we observe. This doesn't only happen when a user is added to the chat but also anytime that a message is sent to the chat, but since the number of users involved in a chat changes when a user is added, that was the interesting case for us. The only case that we cannot distinguish here is when there are no users in a chat versus one user in a chat (i.e. no chat exists or the chat has been created by a user but no one has been added to it yet). In this case, we could use one oracle query to determine which of the two is true. The other operation is used to observe the single user session (passive operation).

Evaluation: Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.21 WithMi 2

A.4.8.2.21.1 Question 012 (SC Space, Intended Not Vulnerable, Answered No)

Response: “For an attacker to determine which file was transferred between two users through a side channel in space, there must be some correlation between the number and sizes of the sent

packets and the file transferred. To ascertain whether this was the case, we began our inquiry by performing some profiling using our Profit black-box network profiling tool. […] Profit's time_partitioned transformation revealed that the sizes of the packets transferred was not deterministic even for the same file. Meaning that transferring the same file twice did not result in the same sequence of packets being sent. This led us to believe that some randomness had been introduced to the code. We inspected the decompiled source code to determine the source of this randomness. The interesting method that we pinpointed was o rg.digitaltip.chatroom.FileTransfer.trasmit(). This method first compressed the file to transfer using one of two compression schemes specified by the sender's command. Then the file was padded with a random number of zeros (with a max padding of 256 bytes). This random number of zeros introduced the non-determinism we observed the profiling. Because of the noise introduced by the zeros, the correlation between the file transferred and the sizes of the packets observed is obscured. Nevertheless, there is still some information that can be gained from the sizes of the packets. For the 5 files we attempted transferring, some of the files were significantly smaller than others after compression and so the limited amount of noise introduced by the extra zeros was not enough to make those files indistinguishable. However, for 3 of the 5 files we experimented with the noise was strong enough that their sequences of packet sizes were overlapping and it was not possible to determine which of the 3 was transferred."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.21.2 Question 018 (AC Space, Intended Vulnerable, Answered Yes)

Response: Vanderbilt repeated the analysis for WithMi 1 question 007 and responded that the challenge program was vulnerable.

Evaluation: The challenge program contains an intended vulnerability in the Huffman compression implementation, which is triggered during compression of certain inputs. The reported unintended vulnerability results in a stack overflow and represents a clear bug in the challenge application. While this is noted, this response is given a post-engagement analysis ruling of incorrect due to the fact that the observed memory resource usage is approximately 84 MB, less than the required resource usage of 2000 MB.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.22 WithMi 3

A.4.8.2.22.1 Question 026 (SC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt performed the same analysis as for PowerBroker 4 question 013. They added that "It turns out that the probability of an extra reduction being performed is proportional to how close g is to the modulus (which recall is either p or q). Thus, as g approaches either p or q from below, the number of extra reductions performed increases and hence so does the execution time. At exact multiples of p or q the runtime greatly falls. This timing information allows us to determine how close g is to one of the prime factors and we can use this information to determine p and q. This analysis would allow us to impersonate a user if we were able to reliably choose our value of g. However, the decrypt function complicates this by first multiplying g by some random value r (an inverse multiplication is later performed in order to ensure that decrypt function properly). Because of this, the value sent in to the exponentiate function cannot be controlled. This destroys our ability to exploit the side channel described

above." Vanderbilt performed further analysis of the session key generation. They added that "This masterSecret is computed in the generateMasterSecret method which calls a circularPow function. The circularPow function is called on the public key of the first user being the base, the private key of the second user being the exponent and the fixed modulus. This function has some time side channel in it exposing the binary structure of the exponent/privateKey (for instance if the private key starts with the chain of 0s vs. chain of 1s (in the same number of bits on) it computes faster) […] The circularPow function also relies on fastMultiply and standardMultiply functions, with the standardMultiply having some random noise added to it. If an attacker had a way to interfere with the communication once the first packages are exchanged and the masterSecrets are built, and if observing the time taken on construction of the masterSecret, along with the information on KeyExchangeServer parameters used – he would be able to get the masterSecret, thus the session key, and continue the conversation using the session key of a victim user. However, from the setting of the problem it seems that interfering in the middle of the communication would not be feasible for an attacker given that he/she is limited to act within the operations supported in the application."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.22.2 Question 030 (AC Space, Intended Vulnerable, Answered Yes)

Response: "The Janalyzer marks several loops as suspicious. One of them, which could be relevant to sizes of written files, is in SpiffyUnpacking.reconstitute(). This method implements a decompression algorithm where repeatedly a byte is read, followed by an integer specifying how often this byte is repeated. […] There is a vulnerability in this loop, because the condition of the inner while loop is $num != 0$, and not $num >= 0$. Because of this, we can write an enormous file by sending only 5 bytes: one byte with random data, then the integer -1. […]"

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.23 WithMi 4

A.4.8.2.23.1 Question 010 (SC Space, Intended Vulnerable, Answered Yes)

Response: "For an attacker to determine which file was transferred between two users through a side channel in space, there must be some correlation between the number and sizes of the sent packets and the file transferred. To ascertain whether this was the case, we began our inquiry by performing some profiling using our Profit black-box network profiling tool. Profit's time_partitioned transformation revealed that though there is randomness is the last (and sometimes second to last) packets associated with the file transferred, the sequence of the first packets transferred for a particular file was both deterministic and characteristic of that file. This led us to believe that there was a potential side channel present. We inspected the decompiled source code to verify if this was the case. The interesting method that we pinpointed was com.digitalpoint.togethertalk.FileTransfer.trasmit(). In this method, the file to be transferred in separated into 256 byte chunks. Each chunk is compressed according to a compression scheme specified by the user's send command (the two options are ZLIB or SNEAKER). Then each chunk is additionally padded by a random amount to a max value of 256 bytes. At first glance, it appears that this random padding would result in non-determinism in the entire file transfer sequence but we saw through profiling that this was not the case.

Inspecting the source code, we realized that the reason for this is that the offset of each message was set so that the additional chunks from the extra padding was ignored. This meant that the sizes of the initial packets in the sequence would be proportional to the sizes of those chunks after compression. The only packet whose size would be non-deterministic would be the last one (as observed). Additionally, we saw in the source code that an extra packet might be randomly sent after the rest of the file was transferred. This explained why there was sometimes an extra packet involved in the transfer. Nevertheless, the sizes of the first packets in the file transfer correlate with the file being transferred. This allows an attacker with knowledge of the contents of the file directory to determine which file was transferred. We verified through profiling that this vulnerability was present regardless of whether SNEAKER or ZLIB compression was used."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.23.2 Question 031 (SC Time, Intended Vulnerable, Answered Yes)

Response: "We began our analysis with network profiling using our black-box network profiling tool Profit. […] We used Profit's time_partitioned transformation to observe the packets sent for each phase. This allowed us to decompose the entire set of interactions into the timing of the packets sent during each reconnect and those sent during each connect. We discovered that there is an observable difference in the pattern of these timing sequences, in particular that in the last chunk of packets sent. This timing difference seemed very consistent and we think we can distinguish between the two cases at least 80% of the time. […] Observing the source code, we found that when two users are connecting for the first time, a file is written to with this connection information. This could explain the timing observations we made using using Profit. We believe this analysis can be automated at the source code level through a control-flow graph level analysis, that will, for each modular component of the code (i.e. a loop, branch, etc) compute a symbolic cost expression that gives all possible values that the component might take in terms of the secret variable and public input. A satisfiability solver such as z3 can be used in order to determine this cost expression might vary in terms of the secret (public inputs must match). We currently have a working prototype of this tool which we can run on a limited way for the source code of this problem which suggests that there might be a timing difference between the two cases."

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.23.3 Question 041 (SC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt performed the same analysis as for PowerBroker 4 question 13 and WithMi 3 question 26 and concluded that the challenge application was not vulnerable.

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.24 WithMi 5

A.4.8.2.24.1 Question 027 (SC Time, Intended Not Vulnerable, Answered No)

Response: Vanderbilt performed the same analysis as for WithMi 4 question 031 and concluded that the challenge program was not vulnerable.

Evaluation: Vanderbilt correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.8.2.24.2 Question 046 (SC Space, Intended Not Vulnerable, Answered Yes)

Response: "[…] we note that there are 3 different types of actions that can change the number of users involved in a chat -- when a chat is created (0 to 1 user), when a new user is added to a chat and when a user disconnects from the application. We consider the "number of users involved in a chat" to be the sum of the user who started the chat and all unique users added to the chat. We began our inquiry by performing some black-box profiling of this variant of WithMi using our tool Profit. […] Profit's space partitioned transformation revealed that, when a new user was added to the chat, a message informing other members of the chat of the new addition would be sent out. For example, if there were three users in a chat and a fourth was added by one of them, then the other two members (as well as the inviter) would receive messages saying that the fourth was added. Therefore, there is a relation between the number of packets sent out when a new user is added and the number of users in the chat. We noticed that if the same user was invited twice to the chat, then there would not be an extra message sent twice for that user. Once we looked into the code, we realized this was due to a dictionary being used to store the relationship between users and the chats they are a part of. We looked into the source code to determine the cause of this behavior and we discovered that when a message is sent to a chat, the method com.techtip.chatbox.DrobBy.trasmitMessage() is called. This method actually iterates through all members of the chat and sends a message to each. This explains the behavior we observe. This doesn't only happen when a user is added to the chat but also anytime that a message is sent to the chat, but since the number of users involved in a chat changes when a user is added, that was the interesting case for us. The only case that we cannot distinguish here is when there are no users in a chat versus one user in a chat (i.e. no chat exists or the chat has been created by a user but no one has been added to it yet). In this case, we could use one oracle query to determine which of the two is true. The other operation is used to observe the single user session (passive operation). We were also curious as to whether or not the size of the packets could leak information about the number of users involved in the chat. While profiling we noticed that the size of the packets sent when two users was involved in the chat was different than that when three users was involved but that the sizes were very close! We discovered that in com.techntip.chatbox.MessageUtils there is a method called formForumStateMsgBuilder which considers by how many users a chat is short of the max of 4 and for each of those users, it adds a null public identity to the message being built. We suspect this may have been an attempt to hide a side channel relating the packet sizes to the number of users in a chat, but it does not impact the space side-channel described above that exploits the number of packets sent."

Evaluation: The challenge program does not contain an intended vulnerability. Vanderbilt correctly noted that padding is added to mask the number of users being leaked in the chat state messages. Vanderbilt claims that the number of packets sent rather than the size of the packets sent leak information about the number of users involved in the chat. The challenge question does not place further restrictions on the chats of the victim user. As a result, the user could be engaged in chats with two other users, and the number of chat messages would not be sufficient to disambiguate whether the user is in one chat with the two other users or in separate chats with each user.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.25 WithMi 6

A.4.8.2.25.1 Question 048 (AC Space Intended Not Vulnerable, Answered Yes)

Response: Vanderbilt performed the same analysis as WithMi 1 question 007 and WithMi 2 question 018.

Evaluation: The application does not contain an intended vulnerability. The reported unintended vulnerability results in a stack overflow and represents a clear bug in the challenge application. While this is noted, this response is given a post-engagement analysis ruling of incorrect due to the fact that the observed memory resource usage is approximately 84 MB, less than the required resource usage of 2000 MB.

Post-Engagement Analysis Ruling: Incorrect

A.4.8.2.25.2 Question 054 (AC Time, Unintended Vulnerable, Answered No)

Response: Vanderbilt investigated recursive and suspicious loop structures flagged by JAnalyzer. Vanderbilt added that: "There are no time-related vulnerabilities in this program. The number of connected users is limited to 2 and the number of previous users is also limited to 2. Although the user name is not limited in length, the max transmitted characters would cap it off below 10,000, which is not enough to cause a significant delay. The chatname is limited to 20 characters, the files are limited to those contained in the data directory, which are all less than 1 kB and the list of files is also only 5. Also, the list of chats cannot be made to be long enough with a 10,000-byte budget to cause a user problems when they are added to a chat and receive all of the old posts. There are too many constraints for a time vulnerability."

Evaluation: GrammaTech and Invincea identified unintended vulnerabilities.

Post-Engagement Analysis Ruling: Incorrect

A.4.9 Invincea (Control Team)

A.4.9.1 Invincea Overview

Invincea answered 39 questions with 92% accuracy. In E4, Invincea had the highest accuracy and the highest TPR against the set of all questions. Invincea's accuracy increased by 18 percentage points from E2 and 25 percentage points from E3.

Table A-79: Engagement 4 Invincea Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	7	6	86	6	86
SC Time	12	10	83	10	83
SC Space/ Time	0	0	NA	0	NA
AC in Space	8	8	100	8	100
AC in Time	12	12	100	12	100
Total	**39**	**36**	**92**	**36**	**92**

Table A-80: Engagement 4 Invincea Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	28	27	**96**	27	**96**
Not Vulnerable	11	9	**82**	9	**82**
Yes Answer	29	27	**93**	27	**93**
No Answer	10	9	**90**	9	**90**

A.4.9.2 Invincea Specific Responses

A.4.9.2.1 Airplan 1

A.4.9.2.1.1 Question 020 (AC Time, Intended Vulnerable, Answered Yes)

Response: "Yes there is an algorithmic complexity vulnerability in time that would cause the real runtime of the program (on the reference platform) to exceed the resource usage limit of 500 seconds, given the budget of 25,000 bytes. This is done by having one user log in, upload the example route map of 500 airports, and then repeatedly request an adjacency matrix. The requests are serviced in a queue and requesting it once every second causes a bottleneck because the server is still processing another matrix request. Doing this approximately 40 times breaks the resource usage limit."

Evaluation: The challenge program contains a vulnerability in the shortest path implementation. It uses an A* search with an inconsistent heuristic. Invincea identified an unintended vulnerability. They observed that after uploading a route map of 500 airports and submitting repeated requests for the adjacency matrix, the resulting time to process all requests allows the final request to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.2 Airplan 2

A.4.9.2.2.1 Question 037 (AC Time, Intended Vulnerable, Answered Yes)

Response: Invincea provided the same response as Airplan 1 question 020.

Evaluation: The challenge program contains a vulnerability in the sort implementation. The worst-case is triggered when the list of items to be triggered is divisible by 6. Invincea identified an unintended vulnerability. They observed that after uploading a route map of 500 airports and submitting repeated requests for the adjacency matrix, the resulting time to process all requests allows the final request to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.2.2 Question 038 (SC Space, Intended Vulnerable, Answered Yes)

Response: "The posted file will not reliably give us the number of airports, since the file a) can be in multiple formats, including txt, json, and xml, and b) can contain information about both the airports and the routes between them. These factors can make the posted file's size variable for the same number of airports, which eliminates it from consideration. We need information that provides us the airport information in a consistent format and that divorces the route information from the airports or in some way aggregates that information so that its size-consistent for every airport. The passenger capacity matrix gives us both of these things. This

matrix tells us how many passengers can fly from an airport to each other airport. The Java class that creates the matrix is FlightMatrixManager.java."

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.3 Airplan 3

A.4.9.2.3.1 Question 002 (SC Space, Intended Vulnerable, Answered No)

Response: "No, there is no side-channel in space such that the attacker can determine whether the route map uploaded by another user is strongly connected.

The first relevant secret is whether or not the graph is fully connected. As part of the upload process, the airplan server sends a packet containing various properties of the uploaded graph to the client. This observable packet contains a string for each property (whether the graph is connected, bipartite, regular, etc). For each value of each property a different string is included in the packet (eg, if the graph is connected, the server will send `Fully Connected`; if not connected, `Not Fully Connected`).

These differences in length cannot be exploited to leak the connectedness of the graph for two reasons. First, the 10 properties reported by the server result in ~12000 possible combinations of properties which result in many distinct combinations of properties with identical packet sizes. For example, the difference in length between the strings for a connected graph and a non-connected graph is 4, but so is the difference for bipartite and non-bipartite graphs. So, all else being equal, it is impossible to use packet size to distinguish graphs that are bipartite and non-connected from graphs that are non-bipartite and connected, therefore preventing differentiation at a 99% level of accuracy.

Secondly, and more importantly, the name of the graph, which is chosen by the user, is included in the packet containing the various graph properties, and so the lengths of the property strings are obfuscated by the length added by the graph name. The length of the graph name cannot be leaked because it is sent to the server along with the graph itself, so all the attacker has is the combined length of graph name and graph file, which cannot be separated without examining the contents of the packets sent, which the attacker cannot do.

Considering the allowed active operations, we note that the attacker could determine if the graph is connected by querying the server for the properties of the uploaded graph. The response to this query would include whether or not the server considered the graph connected, thus trivially providing the answer. However, in order to query for a graph's properties one must request that graph by ID. Graph IDs are assigned randomly (the random number generation process is secure) by the server when the graph is uploaded, and therefore cannot be predicted by the attacker. While the graph ID is included in several responses the server sends to the client as part of the upload process. This value is not leaked because there is no identifiable correlation between the value of the ID and the length of the packets sent by the server. The length of the ID could be leaked, but not its value."

Evaluation: Invincea did not observe that this challenge program displays route map properties in such a way that the size of the packet reveals whether the map is connected, by formatting each of the properties with padding to a certain size. This size is initially set to be large enough to contain all the properties, the longest of which is the string for a disconnected graph.

However, when formatting the properties, there is a check to ensure this space is sufficient; this check is implemented with a '<' where there should be a '<='. If the space is deemed insufficient, the space used for each subsequent property is doubled -- this only happens in the case of a disconnected graph, thus making the properties packet detectably larger in the case of a disconnected graph. There exist fixed sizes, s_d and s_c, such that the properties packet for disconnected (respectively, connected) maps is always of size s_d (respectively, s_c).

Post-Engagement Analysis Ruling: Incorrect

A.4.9.2.3.2 Question 024 (AC Time, Intended Vulnerable, Answered Yes)

Response: Invincea provided the same response as for Airplan 1 question 020 and for Airplan 2 question 037.

Evaluation: The challenge contains a vulnerable implementation of Ford-Fulkerson. Invincea identified an unintended vulnerability. They observed that after uploading a route map of 500 airports and submitting repeated requests for the adjacency matrix, the resulting time to process all requests allows the final request to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.4 Airplan 4

A.4.9.2.4.1 Question 022 (AC Time, Intended Vulnerable, Answered Yes)

Response: Invincea provided the same response as for Airplan 1 question 020, Airplan 2 question 037, and Airplan 3 question 024.

Evaluation: The challenge uses a vulnerable implementation of Dijkstra's to calculate the shortest path. Invincea identified an unintended vulnerability. They observed that after uploading a route map of 500 airports and submitting repeated requests for the adjacency matrix, the resulting time to process all requests allows the final request to exceed the resource usage limit.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.5 Airplan 5

A.4.9.2.5.1 Question 052 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Yes, there is an algorithmic complexity vulnerability in space that causes the challenge program's memory usage to exceed the resource usage limit of 1024 MB. The program's XML parsing code contained in com.roboticcusp.kxml2.io.KXmlGrabber (specifically the next() function), is susceptible to the "billion laughs attack," also known as an XML bomb (https://en.wikipedia.org/wiki/Billion_laughs). The parsing code provides no protection against increasing the length of nested XML entities exponentially, so the example will allocate several gigabytes of characters into memory. I have attached a screenshot of JProfiler showing the memory usage exceeding 1024 MB before the JVM throws an OutOfMemoryException."

Evaluation: Invincea correctly identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.6 BidPal 1

A.4.9.2.6.1 Question 033 (AC Time, Unintended Vulnerable, Answered Yes)

Response: "The bidpal_1 application does not verify a user's BidCommitmentData before using it to create ExchangeData. In the ExchangeData class, make() enters a while loop that is only terminated when verifySpacing(this.array, this.p) returns true. One of the first checks in verifySpacing() is to return false if any of the entries in the array equal 0. In the array passed to verifySpacing(), this.array[0] is equal to the decryption of commit.takeSharedValue() mod p. Note that if commit.takeSharedValue() is 0, then this.array[0] = 0 regardless of the value of p chosen. In this scenario, verifySpacing will always return false, and make() will be stuck in an infinite loop.

By bidding on the same auction as other users, but sending BidCommitmentData with a sharedValue of 0 instead of an honest value, an attacker can cause any users who have bid on the auction, or who later bid on the auction, to be stuck in an infinite loop. In particular, having issued a bid commitment for an auction, an attacker will receive a bid receipt from all connected users, but will not receive the expected ExchangeData from users who have bid on the same auction within 1200 seconds of sending the BidCommitmentData. In fact, the ExchangeData will never arrive because the bidpal_1 application for affected users will be stuck in an infinite loop and unable to perform any further operations without being restarted.

We created a malicious BidDivulgeData.class file with code that sets the BidCommitmentData sharedValue to BigInteger.ZERO in fetchCommitmentData() and patched it into a new malicious bidpal_1_attacker.jar. Note that this is in scope since only the attacker is running the modified program. All other participants in the network environment are running the original, unmodified bidpal_1.jar application."

Evaluation: The challenge does not contain an intended vulnerability. Invincea identified an unintended vulnerability in the bid commitment process.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.7 BidPal 2

A.4.9.2.7.1 Question 044 (AC Time, Intended Vulnerable, Answered Yes)

Response: Invincea provided the same response as for BidPal 1 question 033.

Evaluation: The challenge contains a vulnerability in the auction protocol. Provided a particular username and auctionID, the application generates a checksum that will continuously fail. Invincea identified an unintended vulnerability in the bid commitment process.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.7.2 Question 057 (SC Time, Intended Vulnerable, Answered Yes)

Response: "When user A receives a bid commitment message from user B about auction X, if user A has also bid on auction X, it creates a bid comparison message and sends it to user B. When the user creates a bid comparison message, it makes a call to org.techpoint.sale.messagedata.BidComparisonData.generate(), which contains side channel vulnerable code. The generate() method iterates over all 501 possible bid amounts and runs org.techpoint.sale.messagedata.Deltafier.getDiff() every iteration. For all values less than or equal to the user's bid, getDiff() is run in mode 0, and for all values greater than the user's bid

getDiff() is run in mode -1. Running getDiff() in mode 0 is a fairly inexpensive operation, while running getDiff() in mode -1 executes a much more complicated and time-consuming series of operations. Therefore, the lower a user's bid is, the more times generate() will run getDiff() in mode -1, and the more time it will take to respond to the other user's bid commitment. If an attacker wanted to find out a user's bid, they could simply send a bid commitment to the user for the auction in question, and measure how much time it takes the user to send back a response."

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.8 Collab

A.4.9.2.8.1 Question 028 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "When adding events to a Collab server, a scheduling sandbox is created which does not affect the production event store. All events, including any audit events if they exist, are added to this sandbox (of class collab.collab.SchedulingSandbox).It is worth noting that there is a maximum of one auditid per user, due to lines 319-321 of collab.collab.CollabServer:

```
Integer audid = (Integer)this.auditids.get(sdata.username);
if (audid != null) {
        sbox.add(audid.intValue(), new AuditorData());}
```

This adds at most one audit ID to the ADD sandbox, so in the exploit scenario outlined below there is no need to consider the possibility that more than one audit event exists in the sandbox tree. The sandbox stores event IDs in a modified B-tree of order 9. When a B-tree DataNode becomes full (that is, when it has 8 elements and a 9th is added), the server performs a complex action to split the DataNode into two sub-nodes, and replaces the DataNode with a MedianNode.

The program attempts to disguise the significant extra time that this split operation takes by performing a random number (between 86 and 96) of line writes to a log file (performed by the fileWrite method in collab.collab.CollabServer) if no split occurred during an add operation, but splits still appear to consistently take longer than non-split adds.

Because it can be observed whether a split occurs when adding any event to the sandbox, and audit events behave as any other event would in a DataNode (they just do not show up in a SEARCH), it is possible to keep track of the complete B-tree without the audit node, and then add crafted events to fill up each DataNode, and observe if a split occurs before it would be expected to if the DataNode did not contain an audit node. […]"

Evaluation: The observable timing distribution for split vs non-split while not perfectly overlapped are sufficiently overlapped that it is not believed this can be fully resolved.

Post-Engagement Analysis Ruling: Incorrect

A.4.9.2.8.2 Question 049 (SC Space, Intended Not Vulnerable, Answered No)

Response: Invincea investigated the LOGIN, ADD, SEARCH, DONE, UNDO, and QUIT operations and concluded that "we found no instances in the code base where the presence or absence of an audit eventID would alter the server response to a user command. We conclude that there is no side channel in space from which the user 'picard' can discover an eventID relating to one of his audits within 3000 operations."

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.9 Linear Algebra Platform

A.4.9.2.9.1 Question 034 (AC Space, Intended Vulnerable, Answered Yes)

Response: "When passed a request for a `ShortestPath` computation, the program loads two matrices given by the user. When loading the second matrix (`com.example.linalg.external.operations.ShortestPathOperation`, line 30), which is supposed to be a vector giving the indices of the nodes to calculate the shortest paths from, the function `com.example.linalg.internal.util.MatrixSerializer.readMatrixFromCSV()` is called with `enforceSize = false`. This means that the numbers of rows and columns in the matrix, as declared by the user, are not bounds-checked. Thus, on line 65 inside that function (`double[][] matrix = new double[numRows][numCols]`), the program will attempt to allocate a square array of `double`s with size determined by the number of rows and columns the user declared the matrix to have. By declaring the matrix to be very large (eg 250,000 rows by 1,000 columns) the user can cause the program to allocate correspondingly large amounts of memory. As the memory allocated for these `double` arrays does not seem to be garbage-collected between requests, multiple requests with matrices with very large declared sizes can be made and the memory usage of the program pushed over 6000 MB. (Allocating over 6000 MB in a single request does not seem to be possible, due to limitations in the JVM's use of `malloc`, and so the required 6000 MB must be built up over several requests). As each of the 4 requests sent by the proof-of-concept is only a few hundred bytes, this attack is easily executable within the input budget of 15,250,000 bytes."

Evaluation: Invincea correctly identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.9.2 Question 050 (AC Time, Intended Vulnerable, Answered Yes)

Response: "When passed a request for a `Multiply` computation the program loads two matrices. While these matrices are intended to be square, the checks in `com.example.linalg.external.operations.MultiplyOperation` and `com.example.linalg.util.MatrixSerializer` do not enforce this property. Instead, the code requires that both matrices given have the same dimensions. Thus, rectangular matrices can be passed into the multiplication routine. If two rectangular matrices with more rows than columns are given, the multiplication operation takes longer than when an equivalent quantity of payload data is used to send two square matrices. This is believed to be due to the different patterns of memory interaction induced by the different shape of the matrices. In order to further maximize the time taken all values of the matrices are chosen such that their representations as IEEE 754 doubles have all bits of their mantissas set to `1`, so as to maximize the time required for each multiplication. This pessimization empirically seems to increase the time taken by about 50-100%. Note that in some circumstances sending the proof-of-concept matrices included below may result in the multiply operation taking what appears to be an unbounded amount of time. This is believed to be due to a race condition/logic error in the RPC infrastructure of the program."

Evaluation: The challenge contains a vulnerability in the task distribution for matrix multiplication operations. Invincea identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.10 Malware Analyzer

A.4.9.2.10.1 Question 058 (AC Time, Unintended Vulnerable, Answered Yes)

Response: "Yes, there is an algorithmic complexity vulnerability in time that would cause the real runtime of the challenge program to exceed the resource usage limit of 30 seconds. This is due to a lack of input sanitization in the handling of a "PUT" request in the com.ainfosec.MalwareAnalyzer.Server.serve function within malware_analyzer.jar. When the com.ainfosec.MalwareAnalyzer.Server class receives a PUT HTTP request, it allocates a buffer that is defined by the size of the user-specified "content-length" field - this is done on line 496 of the com.ainfosec.MalwareAnalyzer.Server file. It then reads data from the request into this buffer and creates an MD5 sum of that buffer. Specifying a large "content-length" field, will cause a large delay, as the program is constantly updating an MD5 hash of a buffer that is very large. For the purposes of this proof of concept, I specified the "content-length" as 1 GB. An attacker could increase this more and more and the time to process it would increase linearly. A "content-length" of 1 GB takes over 4 minutes to process on the NUC."

Evaluation: The challenge does not contain an intended vulnerability. Invincea identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.11 PowerBroker 1

A.4.9.2.11.1 Question 019 (AC Space, Intended Vulnerable, Answered Yes)

Response: "All individual inputs are validated for all input files. However, in the case for the capacity field the total capacity is just a total of all generators. In the exploit input file "profile_woa_seller.json" there are two generators. The first has 9 digits of 9's and the second has 6 digits of 9's. In this program, the data structure used to hold the total value of generated power capacity […] The above code does not check to see if the total will grow out of bounds before adding the next generation amount. This creates an unbounded situation and makes an unlimited amount of sell actions. The list of unbounded sell actions is written to the output file winownauction_la.out which makes it grow to greater than 74mb which is larger than the output budget of 1mb."

Evaluation: The challenge contains an intended vulnerability during connection establishment. An attacker can connect to a user and disconnect before the connection is fully established. The user will continuously attempt to re-connect to the attacker logging each attempt. Invincea identified an unintended vulnerability in the storage of the generated power capacity.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.11.2 Question 040 (SC Time, Intended Vulnerable, Answered Yes)

Response: "This is because the number of iterations through the last for loop in the generate function in the ShareData class (included below) is dependent on the myOffer argument, and produces a passively persistent time impact depending on the value of myOffer. An exploit is included which runs the vulnerable code, which comes in the form of 3 modified profiles (i.e. profile1.json, profile2.json, profile3.json) to create the conditions to run the problem (1 seller, 2 buyers). One of the buyers was set to always win the bid and the other was the target. There is

one specific TCP transaction that can tell someone just looking at the data what the delta of the bid is. […] The only assumption is that the attacker knows its own bid. From there the TCP transaction time where Detroit sends its bid comparison to NYC and then the other responds will tell the attacker the delta in dollars Detroit is from the NYC bid. The other basic assumption is that a bid can't be less than the reserve posted bid. […]"

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.12 PowerBroker 2

A.4.9.2.12.1 Question 015 (SC Time, Intended Not Vulnerable, Answered No)

Response: Invincea repeated the analysis used in PowerBroker 1 question 040 and concluded that the challenge program was not vulnerable.

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.12.2 Question 025 (AC Space, Unintended Vulnerable, Answered Yes)

Response: Invincea provided the same response as for PowerBroker 1 question 019.

Evaluation: The challenge program does not contain an intended vulnerability. Invincea identified an unintended vulnerability in the storage of the generated power capacity.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.13 PowerBroker 4

A.4.9.2.13.1 Question 006 (SC Time, Intended Not Vulnerable, Answered No)

Response: Invincea repeated the analysis for PowerBroker 1 question 040 and PowerBroker 2 question 015 and concluded that the challenge program was not vulnerable.

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.13.2 Question 013 (SC Time, Intended Vulnerable, Answered Yes)

Response: "Yes, there is a timing side-channel that will leak a given user's RSA private key. With their private key and username, an attacker can successfully impersonate that user. The RSA timing side-channel comes from a well-documented issue with certain implementations of RSA decryption that use the Chinese Remainder Theorem and Montgomerty multiplication. The flavor of the attack we use is described in "Remote Timing Attacks are Practical" by Brumley and Boneh (https://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf). First, let's look at the vulnerabe portion of the source code. In the CipherPrivateKey::decrypt() method, we see that decryption is broken into computations modulo p and q, aka the Chinese Remainder Theorem optimization. […] This is important for us, as we will leak the moduli used during these exponentiations. Because the challenge program uses the Chinese Remainder Theorem optimization, those moduli are precisely the secret primes P and Q. For now, assume that $P < Q$. Our attack will leak the secret value of P. From the source code above, we see that MontgomeryMultiplier::exponentiate() performs the modular exponentiation. […] We can see that there is a check as to whether to perform an additional modular reduction in the source code

above. Werner Schindler ("A Timing Attack against RSA with the Chinese Remainder Theorem" https://tls.mbed.org/public/WSchindler-RSA_Timing_Attack.pdf) proved that the probability of additional reductions in a computation ($g^d \ mod \ P$) is approximately $\frac{g \ mod \ P}{2R}$. That is, as g approaches P from below, the number of additional modular reductions increases, and if g is slightly larger than P, then the number of modular reductions drops significantly. […]"

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.14 RSA Commander

A.4.9.2.14.1 Question 039 (AC Time, Intended Vulnerable, Answered Yes)

Response: "Yes, there is an algorithmic complexity vulnerability in time that would cause the real runtime of the challenge program to exceed the resource usage limit. After a client receives a message from another client, it sends a terminating message to the sending client using the same nonce for DES encryption, and using the public key supplied by the sending client. Because the nonce is encrypted using RSA, if the sending client uses a very large nonce (the only restriction is that the nonce be greater than or equal to 0 and that the nonce is less than the modulus), then the sending client can force the receiving client to perform a very large exponentiation (on the order of 10^154^65537) which takes well over two hours to compute. Normally, this exponentiation would be computed using much faster modular arithmetic, but if the counter used by the sending client is 0, then BigInteger's .pow() and .mod() are used, instead of the much more efficient .modPow(). […]"

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.14.2 Question 053 (SC Time, Intended Not Vulnerable, Answered Yes)

Response: "Yes, there is a side channel in time in the challenge program from which a third party can determine the plain text of a message sent between two RSA Commander users. In `stac.parser.OpenSSLRSAPEM$INTEGER`, the function modPow branches depending on whether the pow parameter is represented by a Java BigInteger. This is only true if the underlying integer value in the INTEGER pow is outside the range of a Java built-in Integer -- from -2147483648 to 214748647, inclusive. If pow is represented by a BigInteger, then the program's modPow function will call the BigInteger modPow implementation, which uses a relatively efficient modular exponentiation algorithm. However, if pow is within this Integer range, then the program will separately call pow() and mod(), resulting in a time-consuming pow() calculation. Since RSA encryption works by calculating m ^ e (mod n), where m is the plaintext message, e is the public exponent, and n is the modulus, if the public exponent is within this Integer range, then the encryption call will use the inefficient pow() then mod() algorithm. Since the program always constrains imported RSA keys to a public exponent $3 \leq \ e \leq \ 65537$, this condition will always be satisfied. Therefore, a passive observer can gain information about the plaintext message m being encrypted by observing the time it takes a client to encrypt and send a message (observed on the network as time between two subsequent messages). The time delta between the Handshake response from the receiving client and the corresponding Message from the sending client directly correlates to the size of the message m. In this case, the messages m are the nonce and counter parameters for the subsequent DES encryption. These RSA

encryptions are performed on lines 231 and 232 of stac.communications.packets.RequestPacketParser, in the function encryptSymmetricCipherParameters. Since the nonce is a random Java long, and counter is only a random Java short, the impact of counter upon the time delta is negligible. Therefore, the nonce used for DES encryption can be derived using this timing attack, from which the DES symmetric encryption key can be derived by brute forcing the 32767 possibilities for the counter. Local experimentation confirms that the nonce encryption side-channel is strong enough to be observable over the network. The pow() algorithm has polynomial time complexity $O(\ M(n\ *\ e)\ )\ =\ O(n^2\ *\ e^2)$, where n is the number of digits in the operand and e is the exponent, and so a tenfold increase in the nonce results in a $(2n\ +\ 1)\ *\ e^2$ increase in computation time. In practice, using $e\ =\ 6537$, the difference between computing 20000000^{65537} and 200000000^{65537} is an easily observable 0.3 seconds. […]"

Evaluation: The challenge does not contain an intended vulnerability. Invincea identified a potential unintended vulnerability in the application. The relevant components that contribute to the observable runtime of the application are the encryption of the nonce and counter values. During the encryption of these values, the exponent, a value between 3 and 65,537 and the modulus play a role in the resulting observable runtime. Invincea's response indicates that for values of the exponent between 3 and 100 the side channel is weak. As required by the operational definition, the operational budget must apply to the worst-case secret for a vulnerability to be present. In this instance, the vulnerability is expressed by the ability to determine the value of the nonce, a value between 0 and $2^{64}\ -1$. It is noted that for certain values of the nonce, counter, exponent, and modulus, a side channel is present in the total observable time that be used to determine the value of the nonce. However, given the overlap in the range of the nonce, a value from 0 to 32,767, and counter values, as well as the range of the exponent value, in the worst case we do not believe that the identified side channel is sufficiently strong to allow for the extraction of the secret within the bounds provided in the operational budget.

Post-Engagement Analysis Ruling: Incorrect

A.4.9.2.15 Tour Planner

A.4.9.2.15.1 Question 011 (SC Space, Intended Not Vulnerable, Answered No)

Response: “No, there is not a side channel in space allowing a third party to discover which cities a user wants to visit. When a user uses the "query" script on the reference hardware, a series of 9 packets of interest can be intercepted that provide the entire process of the user submitting the query, and the server giving the response. The query script on the reference hardware negotiates a SSL/TLS algorithm that employs padding to 16 byte intervals. This is important as it does *not* provide us with the granularity we need to use the Oracle queries to widdle down to a solution. When submitting all possible $25 * (24\ choose\ 4)$ queries to the server, we throw all the queries into bins based on the packet length sequences over the wire. This results in 5 bins:

['197', '213', '53', '101', '101', '101', '101', '101', '405'] of size 669

['229', '213', '53', '101', '101', '101', '101', '101', '437'] of size 45

['197', '213', '53', '101', '101', '101', '101', '101', '421'] of size 99321

['213', '213', '53', '101', '101', '101', '101', '101', '421'] of size 91294

['213', '213', '53', '101', '101', '101', '101', '101', '437'] of size 74321

Notice that the first packet length corresponds to the users query to the server, and the rest are the server's progress reports and final response at the end. It is only those first and last packets that differ between possible queries. The observable (packet size) relates to the secret (cities in queries) in that if the query uses longer city names, the first and last packets are longer. Otherwise, if the city names are generally short, then we will see smaller packet sizes.

No matter how many passive operations we choose to observe, for the same query the packet lengths will always be the same. This means there really is only one strategy for this problem: observe a single passive query, and then use the oracle queries to widdle down what the possible query was. Because the bin sizes above are so large, we cannot do this with 2999 oracle queries and 99% accuracy. It is worth noting that when a SSL/TLS algorithm is agreed upon that does not do padding (such as a stream cipher), it is possible to get enough granularity to achieve 99.4% accuracy with 2999 oracle queries. This was proven true on a non-reference platform, using the python "requests" library to make our own queries, rather than the provided query script. For this question, the attacker is a third party, so we assume the other parties would use the original query script provided, which is why we answered no.”

Evaluation: Invincea correctly determined that there was not an intended vulnerability. On a side note, the invariance to order reduces the number of possibilities to $25\ choose\ 5 = 53130$ possibilities. The remainder of the analysis accurately confirms what was observed by Apogee.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.15.2 Question 045 (SC Time, Intended Vulnerable, Answered Yes)

Response: “Yes, there is a side channel in time that answers the question. An exploit script is provided that is meant to be run on the reference hardware. It generates a random tour query, observes traffic and timing information, and simulates use of oracle queries within the operational budget to guess query. When a tour query is submitted, the server sends back

progress reports as it calculates a minimum spanning tree for the tour. For a query of 5 cities, there will be 4 (meaningful) progress reports sent that correspond to each edge added in the tree. The observable here is the time it takes one of the 4 progress reports to be sent, and it relates to the secret, which is what edge (i.e. pair of cities) has been added to the tree, which will always be cities from the query. The relation here is a result of how the minimum spanning tree was programmed - city pairs that are geographically closer will take less time to be added to the tree (and hence less time for a progress report to be sent to the client). So, by recording timings of progress reports for all pairs of cities in a tour query, we can construct a sorted list of city pairs by how long they take to be added to the tree, and can hence take a good guess at which edge (pair of cities) was just added with each progress report we receive. The observables reveal the value of the secret, even in the worst case, because the timings are consistent on the reference hardware. By using our infinite active ops to build a sorted list of progress report timings for pairs of cities, we have a baseline accuracy that is the same for any pairs of cities. The timing differences are significant enough that we can detect them, and have coded an exploit script for the reference hardware that will make use of oracle queries to widdle down the possibilities."

Evaluation: Invincea correctly observed that the edge timings provided by the server serve as a side channel. This side channel was initially not thought to be vulnerable. Invincea produced an exploit that takes advantage of the minute difference in the response time based on the edge directionality. This exploit was demonstrated to satisfy the operational budget for cases determined to be at or near the worst case.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.16 Tweeter

A.4.9.2.16.1 Question 021 (AC Space, Intended Vulnerable, Answered Yes)

Response: "When creating a user, the program also creates an avatar for that user. Avatars are, by default, 128x128, but a parameter can be added to the POST for "/register/create" that specifies the size of the avatar. The maximum allowable size is 512x512. Avatars are memory-resident, so every user created causes the memory footprint of the program to increase. Since the avatar is created by the challenge program, the data representing it does not need to be sent over the network."

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.16.2 Question 029 (AC Time, Intended Vulnerable, Answered Yes)

Response: Invincea provided a working exploit.

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.17 WithMi 1

A.4.9.2.17.1 Question 017 (AC Time, Intended Vulnerable, Answered Yes)

Response: Invincea provided a working exploit. Their response also identified the vulnerability in the Huffman compression scheme.

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.18 WithMi 2

A.4.9.2.18.1 Question 012 (SC Space, Intended Not Vulnerable, Answered No)

Response: "When users use "sendfile", the packet sizes over the wire are exactly the same for all the files given as examples. The length of the TCP streams are mostly the same size as well (just 1 "sometimes" exception). It's worth noting that the example files that are given all have the same size to begin with. If they didn't, it would likely be possible to distinguish (the larger the file, the larger the packet size over the wire, assuming the files are about the same entropy). When users use "sendfilezlib", the packet sizes over the wire appear to follow a uniform random distribution (for the packets actually containing the compressed files). It is possible that with enough samples one could determine the file by comparing it to precomputed distributions for each file, although with the problem requirements of 2 passive looks and 99% accuracy, this seems extremely unlikely, especially given how uniform random they all look. TCP stream sizes are mostly the same, but have some variance...again, not enough to achieve 99% accuracy with 2 looks. Looking through the decompiled code, it appears that there is some random padding added to prevent such a passive sidechannel in space. In FileTransfer.class, all file transfers go through the "transmit" method. In there, the file to be transmitted goes through the following call: […] This method takes a random value between 0 and 256, and appends that many 0's to file to be transmitted, resulting in a uniformly distributed file size over this range. […]"

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.19 WithMi 3

A.4.9.2.19.1 Question 026 (SC Time, Intended Not Vulnerable, Answered No)

Response: "No. When users attempt to connect, they use their RSA private keys to sign initial packets that verify their signature. Then, users generate a one-time use Diffie-Hellman shared secret. They use this shared secret to get the session AES key that is used to encrypt all further traffic for that conversation. Since the Diffie-Hellman keys are generated at random during each new connect, it does not seem feasible to leak Diffie-Hellman secrets with a timing side-channel. The AES routine uses external OpenSSL implementations, so those algorithms are out of scope. The RSA private keys consist of primes p and q, with a fixed public exponent e=65537, and a private exponent d. If we can leak either of the primes, we can recover the entire private key and impersonate the user. In CipherPrivateKey::decrypt(), we see that the RSA implementation uses the Chinese Remainder Theorem optimization where it computes c^d mod p and c^d mod q separately, rather than the usual decryption of c^d mod p*q. This optimization is known to be susceptible to timing side-channel attacks, where one uses additional modular reductions to zero in on one of the secret primes. However, surrounding this decryption, the program makes use of a technique called "RSA blinding", whereby each cipher text is multiplied by a random number r, and the resulting decryption is multiplied by r^(-d) mod pq. Because of this random factor r, we cannot control the values being passed into the modular exponentiation function, and thus cannot provide tailored inputs to detect the side-channel. […] The defense of using RSA blinding is a recommended defense to the side channel as described in work by Brumley and Boneh (https://crypto.stanford.edu/~dabo/papers/ssl-timing.pdf)."

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.19.2 Question 030 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Yes, there is an algorithmic complexity vulnerability in space that causes the challenge program to write a file that exceeds the resource usage limit. SpiffyUnpacking::reconstitute() enforces a maximum allowed decompressed totalSize of 100000000 B, which is less than the resource usage limit of 120 MB (120000000 B). reconstitute() is a run length decoder. It takes compressed files of the form: | 8-bit value | 32-bit run length | and expands the runs. The function logs the run length before writing the bytes, so it's not possible to write, e.g., (MAX_SIZE - 1) B then 2 B to exceed MAX_SIZE. However, the run length is a signed integer. reconstitute() does not check if the value is negative, so it's possible to "write" a negative value of bytes to adjust totalSize. […]"

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.20 WithMi 4

A.4.9.2.20.1 Question 010 (SC Space, Intended Vulnerable, Answered Yes)

Response: "Yes, there is a side channel in space that allows for passive determination of a file sent using 2 passive operations with an accuracy of 99%. For now, this solution applies only to files sent using "sendfile", although it's suspected that even when using "sendfilezlib" this will be possible by using a different fingerprint for each file. Here, the observable is the size of packets sent over the wire, which relates to the secret of which file was sent in that every file generates a unique sequence of packet sizes going over the wire. Assuming we have access to the victim machine to view the file contents (allowed for this question), we can determine these packet sizes as one-time offline work by copying the files and sending them, say, to ourselves in our own session. Then we can look for these sequence of packet sizes to determine the file in 1 operation. […]"

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.20.2 Question 031 (SC Time, Intended Vulnerable, Answered Yes)

Response: "Yes, there is a side channel in time from which an attacker can determine whether two users in the process of connecting have previously connected. During a first connection between client user A and server user B, user A needs to write user B's information to his data/<user>/previous_users.txt file. (User B needs to do the same thing for user A.) This occurs in WithMiConnectionsService::addMemberToFile(), which is only called if the user isn't known by the MemberManager. MemberManager::addMemberToMemberHistory() calls the previously-mentioned function. The client writes the file before it sends a receipt packet to the server, and the write time is noticeable. […]"

Evaluation: Invincea identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.20.3 Question 036 (AC Space, Intended Vulnerable, Answered Yes)

Response: "Yes, there is an algorithmic complexity vulnerability in space that causes the challenge program to write a file that exceeds the resource usage limit.

SneakerWrapper::stretch() enforces a maximum allowed decompressed totalSize of 100000000 B, which is less than the resource usage limit of 120 MB (120000000 B). stretch() is a run length decoder. It takes compressed files of the form: | 8-bit value | 32-bit run length | and expands the runs. The function logs the run length before writing the bytes, so it's not possible to write, e.g., (MAX_SIZE - 1) B then 2 B to exceed MAX_SIZE. However, the run length is a signed integer. stretch() does not check if the value is negative, so it's possible to "write" a negative value of bytes to adjust totalSize. […]"

Evaluation: Some variants of this challenge fail to check the negative value of the run length, for this variant, the intended vulnerability is in overflowing the run length integer. Invincea identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.20.4 Question 041 (SC Time, Intended Not Vulnerable, Answered No)

Response: "No. When two users connect, they use their private RSA keys to sign the initial packets establishing their identity. They then generate a (random) shared secret using Diffie-Hellman keys that are generated for each session. This shared secret is used to establish the session's AES key. Since the Diffie-Hellman keys are only used once per session, and are re-generated for each new session, we do not think they are vulnerable to a timing side-channel attack. AES is performed by external libraries, so it is out of scope. The RSA implementation does not suffer from any well-known timing side-channel attacks. Unlike other STAC problems in this engagement, this implementation does not use the Chinese Remainder Theorem optimization. […] This initially appears like it might have a timing side-channel, since different values of the secret bits of y can induce either a single or double modular multiplication. However, if we look at fastModularMultiply() […] we see that a random multiple of the modulus is summed to the result before modular reduction. Therefore, the runtime of the modular multiplication routine is not predictable, given a tailored input. This blurs any timing side-channel that relies on the number of modular reductions. Without any method of distinguishing runtimes between chosen ciphertexts, it is impossible to craft a timing side-channel to leak the RSA secret key."

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.21 WithMi 5

A.4.9.2.21.1 Question 014 (AC Space, Unintended Vulnerable, Answered Yes)

Response: Invincea provided the same response as for WithMi 4 question 036

Evaluation: Some variants of this challenge fail to check the negative value of the run length, for this variant, there is no intended vulnerability. Invincea identified an unintended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.21.2 Question 027 (SC Time, Intended Not Vulnerable, Answered No)

Response: "No, there is no side channel in time from which an attacker can determine whether two users in the process of connecting have previously connected. During a first connection between client user A and server user B, user A needs to write user B's information to his

data/<user>/previous_users.txt file. (User B needs to do the same thing for user A.) This occurs in WithMiConnectionsService::addCustomerToFile(), which is only called if the user isn't known by the CustomerManager. CustomerManager::addCustomerToCustomerHistory() calls the previously-mentioned function. The client writes the file after it sends a receipt packet to the server, so the write time is not noticeable when analyzing network traffic timing. […]"

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.21.3 Question 046 (SC Space, Intended Not Vulnerable, Answered No)

Response: "[…] the application-level connection is not sufficient information to determine how many users are in a chat: three users who first connect to each other via the command line and then start a three-person chat will have the same number of peaks as three people who first connect to each other via the command line and then have only two people start a chat with each other. Therefore, an attacker must consider the constant-size message sent to everyone in the chat. In WithMi_5, these messages include a list of users in the chat and are padded to be a minimum length, making it difficult to determine the number of users in the list based on size alone (i.e. a message with a list of three users will be approximately the same length as a list with two, assuming the users have ids of similar length). But since these messages are sent out at approximately the same time, it is possible to determine how many users are in the chat based on how many of these messages are sent out at once.

However, WithMi users are not guaranteed to be in only one chat in a single session and an attacker can only record up to two WithMi sessions in order to determine who is in the chat. Imagine that Sally is in a chat with X number of people and that she invites Megan to join her. Sally is also in a chat with Y number of people (where X != Y) and invites Deven to join her. Let's say an attacker is trying to determine the number of people in Sally's chat with Megan and X other users. An attacker recording Sally's traffic will see X constant-sized messages sent in succession and Y constant-sized messages sent in succession, but will not be able to determine which chat has X users and which has Y (an attacker could query the oracle if X is the correct number of users and guess Y if it is not, but Sally could also be in another chat with Z number of users, reducing the usefulness of this method). Is there another user whose traffic could be recorded to help determine the number of users? Recording Megan's traffic could potentially help determine whether X or Y people are in Sally's chat if Megan receives Sally's message before Sally sends the Y messages, then there are X other people in the chat, but there is no guarantee that Megan will receive the message at that time (and will definitely not if Sally invited the new user to the Y chat first). Recording the traffic of any other user in the chat gives way to the same problem as determining if there are X or Y users any other user may be in another chat whose traffic conflicts with the chat with Sally."

Evaluation: Invincea correctly determined that there was not an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.4.9.2.22 WithMi 6

A.4.9.2.22.1 Question 054 (AC Time, Unintended Vulnerable, Answered Yes)

Response: "Yes, there is an algorithmic complexity vulnerability in time for this challenge. The Spiffy compression algorithm uses a trie to encode data values. Each leaf in the trie is a data

value, and during decompression a bitstream is used to traverse the trie. Trie traversal starts at the root, and bits are read from the bitstream to pick the first or second child of the current node until a leaf node is reached. The data value of the output is the value of this leaf. The data format of the compressed file is as follows:

| bits | description |

| 32 | isPadded |

| 1,17 | trie node |

| 32 | length |

| >=length | bitstream |

Trie nodes that are not leafs use 1 0-value bit. Trie nodes that are leafs use 1 1-value bit and a 16-bit data value. length is the number of 16-bit values to decode. Each value normally requires >= 1 bit from the bitstream to resolve. This algorithm is vulnerable if the root node of the trie is a leaf, i.e., the trie only has a single value. In this case, the value requires 0 bits from the bitstream to resolve. The output file will consist of the value stored by the root node repeated length times. The maximum value of length is 2147483647, so it's possible to cause the remote side to create a $2147483647 * 2$ B file with a Spiffy archive that's only 11 B large."

Evaluation: The challenge implements an intended non-vulnerable Huffman compression algorithm. Invincea identified an unintended vulnerability in the challenge application.

Post-Engagement Analysis Ruling: Correct

A.5 Engagement 5

A.5.1 University of Colorado Boulder

A.5.1.1 Colorado Overview

Colorado answered 23 questions in the take-home engagement with the second highest blue-team accuracy of 74%. During the live engagement the team answered 24 questions and increased their accuracy to 83% (6th highest). Colorado had the second smallest increase from take-home accuracy to live accuracy, but in cases where the team changed their answer from the take-home to the live engagement, they were accuracy 100% of the time. The team had the most difficulty with SC Time and SC Space/Time questions in the take-home engagement with 50% and 0% accuracy respectively. Given that there was only 1 SC Space/Time question their performance in this category may not be indicative of their capacity to answer this type of question. The team performed much better on vulnerable than on non-vulnerable questions, with a 13 percentage-point difference in the two during the take-home engagement. During the take-home engagement, on Medpedia Question 009, the team identified the potential for a side channel without the presence of random padding but missed that the random padding has insufficient entropy resulting in a vulnerability. On Stegosaurus Question 003, the team missed that the dependence of the heartbeat thread on a variable controlled by the thread performing the *hide* operation allows for a segmented side channel exploit; however, a broken bound check allows results in a weak side channel. On SimpleVote Question 025, the team identified the intended vulnerability; however, as Vanderbilt demonstrated that a loss of information resulted in the

application being not vulnerable, the answer key was changed to reflect this. During collaboration as with all other teams, Colorado answered this question correctly.

While reviewing the take-home engagement responses for Poker Question 004, this question presented an interesting collaboration opportunity as GrammaTech and Colorado identified a weak side channel and ruled correctly that it was not sufficiently strong. Vanderbilt and Iowa had identified similar potential side channels but ruled them sufficiently strong. The decision point for the presence or absence of a vulnerability was due to whether the random seed used to pad packets could be determined. As the result of collaborations, all teams correctly concluded that there was no way to determine the random seed for the benign user and predict the padding sequences, as separate *Random* objects are used in the benign user and attack threads and the challenge question limits the attacker to a single game with the benign user. This was a highlight of what was expected from the collaborative engagement.

A.5.1.1.1 Take-Home Engagement Scores

Table A-81: Engagement 5 UC Boulder Take Home Engagement Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	5	4	80	3	80
SC Time	4	2	50	2	50
SC Space/ Time	1	0	0	0	0
AC in Space	7	6	86	6	86
AC in Time	6	6	100	6	100
Total	**23**	**18**	**78**	**17**	**74**

Table A-82: Engagement 5 UC Boulder Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	19	16	**84**	15	**79**
Not Vulnerable	4	2	**50**	2	**50**
Yes Answer	18	16	**89**	15	**83**
No Answer	5	2	**40**	2	**40**

A.5.1.1.2 Live Engagement Scores

Table A-83: Engagement 5 UC Boulder Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	5	4	80
SC Time	5	2	40
SC Space/ Time	1	1	100
AC in Space	7	7	100

AC in Time	6	6	100
Total	**24**	**20**	**83**

Table A-84: Engagement 5 UC Boulder Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	20	17	**85**
Not Vulnerable	4	3	**75**
Yes Answer	20	17	**85**
No Answer	4	3	**75**

A.5.1.2 UC Boulder Responses

A.5.1.2.1 Accounting Wizard

A.5.1.2.1.1 Question 011 (SC Time, Unintended Vulnerable, Live: Yes)

Live-Collaborative Response:

"Yes, there exists a side channel vulnerability in time from which an attacker with Employee credentials can discover the project expenditure limit for a project they work on. This attack is closely related to the AC time vulnerability from question 23. More specifically, the attacker is able to control the running time of the expenditure report which as we already knew from question 23 has an exponential worst case running time. Furthermore, the condition that triggers the exponential behavior in the expenditure report depends on the secret. The vulnerable code is inside class BudgetSolver, method getAllowedExpenditures. The branch with condition $adjustedExpenditureLimit \geq nonRequiredExpenditure$ is tainted by the secret and is the one that guards the exponential behavior."

Live-Collaborative Evaluation:

There was a mistake in the packaging of this challenge program as it came with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects' contents or states. It is unclear if the clarification given during the live engagement created ambiguity about the state of the project under attack prior to the attack.

The vulnerability reported by Colorado and Utah, was investigated prior to the engagement and is not believed to be sufficiently strong. There is an SC Time vulnerability that leaks the residual budget. The Project Expenditure Limit is the secret; the Adjusted Expenditure Limit is the Project Expenditure Limit minus the hours and items charged by other users. By combining the time differential for an employee's total hours being under vs over the Project Expenditure Limit with the intended AC Space vulnerability an attacker can cause a sufficient time differential such that a binary search is feasible. The binary search depends on the attacker's ability to set the total hours charged on a project to zero. As the attacker is an employee, they are only able to set their

own total hours charged to zero. As a result, this side channel only leaks the Adjusted Expenditure Limit and not the required Project Expenditure Limit.

Several STAC teams identified an unintended vulnerability during the live collaborative engagement. In the case where an employee has no hours, an attempt to add hours causes an exception to be thrown. In handling the exception, the newly submitted hours are directly compared to the secret Project Expenditure Limit in FileStore.hoursFit() method. This comparison triggers writes to the log file if the newly submitted hours are less than or equal to the secret Project Expenditure Limit. The number of log writes that occurs controls the strength of the side channel. As delivered, the number of log writes is insufficient to distinguish when newly submitted hours exceed versus don't exceed the secret Project Expenditure Limit. However, an intended AC Space vulnerability where verbose logging is triggered can be combined with the addition of more items to a project to increase the number of log writes.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.1.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"No, we believe that current budget (50 kB) is not enough for achieving the desired algorithmic complexity vulnerability in space (1 MB). Throughout the application, there are two ways to cause the challenge program to store a file: fi First, the attacker can craft certain invalid inputs that trigger the error exceptions which will be accumulated to a file called. accounting.log on the server-side. For instance, by constructing the following request of deleting an invalid item: curl -k -L -b myCookieFile.txt -H "Content-Type: application/json" -X DELETE https://serverNuc:4567/employee/deleteItem/NEWPROJECT/P/2/a the system will throw the following exception and then write it to .accounting.log: […] However, since each request consumes a PDU size of around 600 bytes and each exception is about 2.4 KB, our current attack creates a 0.2 MB file and is not able to achieve the target resource usage (1 MB with 50 kB budget), and to achieve the desired resource usage (1 MB), we will need a budget of 250 kB. However, this is indeed suspicious because the attacker can generate arbitrarily large file given enough budget. fi Second, the attacker can create a new item which will essentially be serialized to a file called .store/<HASHCODE> chunk where HASHCODE corresponds to the hashCode of that object. By analyzing the method toBytes in c.b.a.o.BaseObjectMarshaller, we found that the size of the serialized object will be bounded by the number of fields in the object. Since c.b.a.o.Project contains three fields of long type and one field of string type, and c.b.a.o.Hours contains three fields of long type and two fields of string type, the size of the serialized Project or Hours object will be around 60 bytes. To achieve the desired resource usage (1 Mb), the attacker would have to perform nearly 16,000 requests, which is way above the original budget. As a result, the attacker will not achieve the desired resource usage within the given budget."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attempt to switch the local to and unsupported on triggers verbose logging mode that can be used to submit inputs that cause the server to exceed the resource usage limit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“After the collaborative engagement we decided to change our answer. There is an algorithmic complexity vulnerability in space that causes the application to store a file with logical size over 50 KB. The attack first sets the locale to a language that is not supported which has as a result the log-level to be set to TRACE. Next, the attacker can start ordering several items for a project they are part of. This last step will cause the application to log the contents of the entire project after every order (i.e., a quadratic order of inflation in the log file), this happens inside the BaseObject class when the uuid of every item is calculated. After the collaborative discussion, we were able to independently reproduce the exploit (in a more efficient way than the one presented during the engagement). Our attack, which can be found inside folder exploit 14, orders items with a very long name and large quantities/costs in order to minimize the number of application requests.”

Live-Collaborative Evaluation:

Colorado identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.1.3 Question 023 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Yes, there exists an algorithmic complexity vulnerability in time as demonstrated by the scripts in directory exploit 023. Essentially, the attacker can control the running time of the benign request by ordering a set of items, in a specific project, with a combined cost that exceeds the budget of the project.The attack exploits the exponential running time complexity of the method:com.bbn.accounting.budgeting.BudgetSolver.getAffordableLinesThis method identifies a maximal subset of the items that can be ordered with the current budget and is only being called by the benign request when the total cost of the items exceeds the project's budget. In order to determine a subset that can be afforded, getAffordableLines enumerates all possible subsets of the ordered items until it finds one that can be afiorded. The control ow graph of getAffordableLines (see Figure 3) shows the enumeration that is controlled by vectorDec, which performs the subset construction (see Figure 4).The reverse call graph (shown in Figure 5) shows that the getAffordableLines method is accessible from an Employee Account: the EmployeeRoutes represents the spark entry point for Employees; the EmployeeManager.fit method only checks if the project is valid and does not impose any further restrictions on access.Profiling indicates that most of the time is spent in the getSum method, which, according to our cost model has a base cost of more than 6000 ns.”

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Yes, there is a side-channel vulnerability in space from which the attacker can infer the number of budget items in a specific project. An attacker is capable of controlling the size of the file that stores all the details about a project by using the \Order Item", \Delete Item" and \Update Hours" requests. Furthermore, an attacker can obtain a list of pairs, via the \Get File Sizes" action, of the form (a; b), where a is a file size and b is the number of files with size a. Sample scripts that demonstrate this vulnerability are included in folder accounting wizard/exploit 026.More concretely, each file that tracks a single project changes predictably, by 44 bytes, after adding/removing an item or updating the hours of an employee. Therefore, the attacker employee first requests all the file sizes in the system and then adds a single item to the project they are interested in inferring the secret. Next, the attacker can request the file sizes once again and infer which file changed by 44 bytes from their previous request (see Fig. 6). Last, the attacker can infer the secret information, let's say, by solving the following equation:

$$s = (fileSize - 34)/44$$

Equation A-5: Accounting Wizard Challenge File Size Equation

{"fileSizeCounts":[{"size":66,"count":1},{"size":34,"count":1}]}

200 https://localhost:4567/employee/report/files

204 https://localhost:4567/employee/updateHours/NEWPROJECT

{"fileSizeCounts":[{"size":66,"count":1},{"size":69,"count":1},{"size":78,"count":1}]}

200 https://localhost:4567/employee/report/files...”

Take-Home Evaluation:

Colorado identified the intended vulnerability

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.2 BattleBoats 1

A.5.1.2.2.1 Question 006 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Yes, there exists an algorithmic complexity vulnerability in time. The vulnerability is in the method com.cyberpointllc.stac.battleship.StrikeLocater.toRadians. This method takes a double as argument and repeatedly subtracts (or adds) 360.0 until the value is in the range 0-360 (see figure

7 for the control ow graph). The value of the double, thus, directly controls the number of iterations performed. This value is received as a string from the peer (see reverse call graph in figure 8) and passed on as a string without any sanitization to the StrikeLocator constructor, where the string is parsed into a double and passed on, again without any sanitization (see figure 9 for the relevant byte code) to the toRadians method. Thus, the attacker can enter a shoot command like shoot 2 2 45 4E100 and make the peer hang for a long time (much longer than the limit set). In effect, there is a doubly exponential increase in time relative to the representation of the input. An attacker can force this loop to run for a very long time (more than 120s) on a benign client instance by sending a shoot command with a negative height value. Due to buggy sanitization code in ccsb.Competition.fetchStrikeSquares and ccsb.StrikeLocator.StrikeLocator|which do not throw an exception when the height is less than zero, a benign instance always calls the ccsb.StrikeLocater.calculateStrikeTime method with any values it receives from an attacker via a shoot command. Therefore, a simple script as the one in the directory exploit 006 can trigger this algorithmic complexity vulnerability in time."

Take-Home Evaluation:

Colorado identified an unintended vulnerability in the challenge program.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.3 BattleBoats 2

A.5.1.2.3.1 Question 010 (SC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Yes, there exists an algorithm complexity vulnerability in time that allows the attacker to determine the position of their cannon on the opponent's board. We take advantage of a timing side channel that can be observed between the attacker issuing a shoot command and the victim issuing a response. After issuing a shoot command, the attacker will take longer to receive a response if they have fired outside of the game's ocean board, regardless of where the shot goes on their radar board. (Figure 10) shows the difference in timing between the slow and fast mode of this side channel. In order to make use of the above procedure, the attacker must discard the first 10 shoot commands that they issue as these do not present an observable side channel. The attacker must wait longer before they receive a response from the server for these commands, possibly due to JIT compilation not being fully engaged. We can determine the cannon's x-coordinate on the board by the following procedure. Fire half of the board's length along the x-axis of the board. If our side channel indicates that we have fired outside the board, reduce our shot distance by one and fire again. Otherwise increase outshot distance by one and fire again. Continue increasing our decreasing the distance of our shot and until we find the boundary at which our side channel switches modes. This provides us with the x-coordinate of our cannon. We can repeat this process for the y-coordinate. In the worst case, given that the board is 20 x 20,

we will need 30 operations in total. 10 for the side channel to become observable and 10 to determine each coordinate."

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"As before, we conclude there exists a side channel in time that allows the attacker to determine the position of their cannon on the opponent's board. Based on discussions with team members from Northeastern, Vanderbilt and Iowa State, we add more details on the location of the vulnerability in the code, checked by our own tools. The reason we did not find this justification with our tools earlier is that we had not yet provided cost expressions for the wait method. When we add a cost expression to our library method summary for the java.lang.Object.wait method the CodeHawk cost model produces the following cost report where it shows a very high constant cost for the method Competition.isOver(): […] Inspection of this method shows an unconditional waiting on an object for 30 ms. Inspection of the reverse call graph for this method, shown in Figure 10 shows that the method is called in the handling of a shot taken. The method that determines whether the isOver method is called is OeanPanel.pullOceanSquares: if either the x-coordinate or the y-coordinate is outside the board control flows to pc=88 where the isOver method is called (see control ow diagram in Figure 11). Thus, the trigger for the side channel is located in the OceanPanel.pullOceanSquares methods. The CodeHawk cost model for this method shows the following block costs […] To maximize the strength of the side channel that differentiates between a shot taken between inside the board (pc=0, pc=79,pc=135) and outside the board (any other path through the method), judging from the block costs shown above, we should aim for a path that includes the pc=88, 111, 119, and 135 blocks (we should avoid pc=127), that is, the shot should be outside the ocean board, but not outside of the radar board (indicated by the conditional expression tooFar(xSeparation) and tooFar(ySeparation). In fact, based on discussions with the Iowa team, any other path through the control ow graph, significantly weakens the side channel, which probably explains the relatively small-time difference we observed in our original response, which did not take this effect into account. We also run experiments and observe that it takes 138 ms to handle shoots inside ocean board and 170 ms to handle shoots outside ocean board but inside radar. Shoots that go outside of the radar board takes around 155 ms, which confirms the insights from the static analysis."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.3.2 Question 021 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Yes, there exists an algorithmic complexity vulnerability in space. The vulnerability is in the method ccsc.HitLocator.computeHitTime. This method computes the travel time of the ball shot from the cannon. This method tries to compute the time in a loop until it cannot find a better estimation of the travel time with the fixed precision of 1 e-100. In order to find the desired time, the while loop keeps putting the calculated times with their corresponding data to the

HashMap<BigDecimal, TrajectoryData> trajectory until it cannot update the time with the given precision. An attacker can cause a benign instance to inflate this HashMap by sending a few shoot commands with very small height values, as shown in the scripts in in the directory exploit 021. A very small height value could trigger the loop to be executed for many times. Our tool of loopr and pathr have helped us pin point suspicious program locations, loops in particular. It is important to note that ccsc.HitLocator.computeHitTime is reported by both loopr and pathr. In addition, the output of pathr (Figure 12) has much fewer suspicious program locations than the ouput of loopr (Figure 11), because pathr has excluded some loops who are reported by loopr but are linear or don't have any memory costly instructions inside, including method invocation instruction. We manually annotate instructions that are memory costly, such as invocation to java.util.Map.put."

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.4 BraidIt 1

A.5.1.2.4.1 Question 002 (SC Time, Intended Not Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Yes, there is a side-channel vulnerability in time from which the attacker can determine which of the five braids was selected and modified by the victim in round 1 and round 3 of the game.The vulnerability exploited is the fact that in a game with numStrands set to 3 (which is under the attacker's control) the victim can only perform swap operations; the game does not allow any operations that change the length, so the length of the modified braid is always equal to the length of the original one. The length of the modified braid strongly affects the time it takes to check for equivalence, so by controlling the lengths we can force time separation. […]"

Take-Home Evaluation:

The challenge does not contain an intended SC Time vulnerability. The instance described by Colorado can be used for an SC Space vulnerability, but we do not believe there is sufficient separation for it to be used as an SC Time vulnerability. The equivalence check algorithm is entirely within the attacker's control, therefore timing the equivalence check on the attacker's application is out-of-scope as that is not the application under attack.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.1.2.4.2 Question 007 (SC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"Yes, there exists a side-channel vulnerability in space that allows the attacker to determine which two braids were modified. The attack exploits the same vulnerability described in Question 002, that the length of the braid cannot be modified when the number of strands is 3. In this case the strategy is simple: in both rounds the attacker sends five distinct lengths to his opponent. The length of the modified braid directly reveals which braid was modified."

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.5.1.2.4.3 Question 017 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Yes, there is an algorithmic complexity vulnerability in space that causes the target application to store files with combined logical size that exceeds 25 MB. The attack, demonstrated by the scripts inside the folder named exploit 17, exploits the run-time behavior of method isEquivalent in class Plait. Specifically, isEquivalent calls (transitively) methods simplifyOnce and freeSimplify which log a message to a file every time they are executed. In order to force the victim's client to execute these two methods an excessive number of times, we created a rogue client that includes the following modifications"

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.5 BraidIt 2

A.5.1.2.5.1 Question 001 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The attacker sends 5 lengths as f51; 51; 51; 52; 52g in both the rounds (round 1 and round 3). If l is the length of the chosen braid, then the expand3 modification is not possible ifl+2 > 52. This is shown in control ow graph in Figure 23. Similarly, the expand5 modification is not possible if l + 4 > 52. This is also shown in Figure 24. Hence by choosing the 5 braid lengths as f51; 51; 51; 52; 52g, the attacker takes expand3 and expand5 (the only modifications that change the length

of the braid) out of the picture and effectively forces the victim to only apply those modifications that do not change the length of the braid."

Take-Home Evaluation:

Colorado's exploit counts active operations as attacker inputs to the console rather than remote requests sent to the peer under attack. This exploit exceeds the operational budget as connecting and playing the game requires 10 BraidIt requests to the remote endpoint leaving only 2 operations for Oracle queries.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.1.2.5.2 Question 022 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Yes, there is an algorithmic complexity vulnerability that causes the real runtime of the benign request to exceed 300 seconds. The exploit, inside folder exploit 22, is identical to the one presented in section 4.1.3. The only difference in this case is that the vulnerable class is called Braid instead of Plait."

Take-Home Evaluation:

The vulnerability in BraidIt 2 differs slightly from the one in BraidIt 1, but there do exist overlapping exploits.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.6 IBASys

A.5.1.2.6.1 Question 029 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"In this problem, in order to trigger the space vulnerability, we need to find a way to write a lot of logs into the `logs' directory created by the server. The only reachable place for writing logs is in com.ainfosec.ibasys.server.IBASysServer.init, where there is a catch construct that writes the current session into the `logs' directory usingScalr.log. Scalr.log will write session into an XML file called \<session.hash>.xml", and the content will be the buffered data in this session. […] So, in order to write a lot of data into logs, given the budget limit, we need to achieve

threegoals:1. We need to put a lot of data into the current session bufier2. We need to cause an exception to be thrown at position (1) because, before that point, session = null.3. We need to change the session hash such that each log goes into a different XML file. Goal 1 is easy, because no matter what we uploaded, as long as it goes to the same session, itis going to be buffered in that session, even if we have completed a whole sequence of uploads corresponding to an image passcode. Goal 2 can be a achieved by an upload with data = all zeros. When uploaded, it will case an IndexOutOfBound exception to be thrown at position (3) in parsePacketPart (shown below),because total = 0 for this upload. […]Goal 3 can be achieved by changing the user name data in the upload headers. Because the session hash is implemented as the hash of the session's user name, we just need to find a way to modify a session's value without losing its buffered data. To achieve this, we just need to change the user name filed in the upload headers but maintain the same aes key, and the session name will be overwritten at position (1) in the above code. […]Here we set the variable buffer in IBASysServer.init() to be user-tainted source. We set array indexing as the taint sink to find potential IndexOutOfBoundException. The ow shows that an array index in LoginManager.parsePacketPart() depends on the user input, which helps us craft the exploit mentioned before."

Take-Home Evaluation:

Colorado correctly determined that the challenge program is vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.7 Medpedia

A.5.1.2.7.1 Question 009 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Albeit, there is a side channel vulnerability in space. The size of the HTTP request from the benign user is a linear function of the length of the requested article's title. The size of the HTTP response to the benign user is determined by the requested article's content size. We have collected the HTTP request and response size for each article in file sizes.txt underexploit 009 directory. If the attacker is able to get the exact HTTP request and response size, she will be able to pinpoint exactly which article is requested. To mitigate the aforementioned problem, the server adds a random padding of size 32 to 2048to each response as a custom X-Padding response header. Now, for each article there could be2016 different possibilities of response sizes, which makes the exploit harder. Nevertheless, we wrote a script check.py which enumerates all possible articles and paddings, and group them according to their corresponding request and response sizes. It turns out that the largest group we can get contains 194 possible articles, which means that an attacker could still identify the secret article by making 194 oracle queries at most even in the presence of random padding. Unfortunately, this application uses

HTTPS protocol instead of HTTP protocol. As a result, the underlying TLS layer will encrypt, MAC, and chunk the HTTP payloads from the application layer. Under the setting of the reference environment, the connection will follow TLSv1.0protocol and use TLS DHE RSA WITH AES 128 CBC SHA as its cipher suite. The AES algorithm will always pad the HTTP messages to 16-byte boundary. This padding further obscures the raw HTTP request and response sizes. If we take TLS into account and measure the raw packet sizes instead of the application-level payload sizes, we are no longer able to uniquely determine the secret article under the given budget: simply apply the padding to all the request sizes insizes.txt and re-run check.py again would show that the number of required oracle queries rise from 194 to 2269, which is clearly above the budget."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The random call does not have sufficient entropy. The attacker can determine the possible paddings using active operations and use the relationship of html page size to article to determine the secret.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"During the discussion we are informed that the length of the random paddings can only be picked from one of at most 8 possibilities instead of 2016 possibilities during a single run of the server. Moreover, one of the possibility is actually fixed, so it all boils down to 7 possibilities. Originally, we noticed the high repetition rate of random padding length but failed to notice that the total number of possibilities is bounded. It turns out that the random generator used by the server is shared among all sessions, and the attacker can simply query the server himself and collect what random padding sizes that will be sent to the victims. Collecting all 7 possible random numbers is essentially a Coupon Collector's problem. The tail bound can be derived from the following equality, where n is the number of coupons and T is the number of trials:

$$Pr(T > n \log n + cn) < e^{-c}$$

Equation A-6: Medpedia Challenge Coupon Collectors Problem Equation

For this particular problem, n would be 7, which means that picking c = 3 (i.e. around 41 trials) will guarantee a 95% success rate of collecting all numbers. Reducing the number of possible padding sizes drastically changes the game, as now all noises in the packet sizes, namely random padding (8 possibilities), AES padding (16 possibilities) and HTML chunking (4 possibilities), are all small enough to be covered by oracle queries separately. The only question remains is whether the budget is enough for all noises combined."

Live-Collaborative Evaluation:

Colorado identified the intended vulnerability through collaboration; however, the team stated that the number of unique paddings is 7 for each server instance. Further investigation is required to confirm the reduction from 8 to 7 paddings, but this doesn't affect the post-engagement evaluation.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.8 Poker

A.5.1.2.8.1 Question 004 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"No there is no strategy the attacker can use to infer his opponent's hole cards within 175 operations while maintaining a success rate of 95%. […] Furthermore, we make the following assumption about the environment on which the application is executed: The random number generation algorithm used by the application, i.e. ThreadLocalRandom.nextInt(), is properly implemented and generates random numbers in the given range uniformly at random. The attacker cannot predict the numbers that will be generated next. […] Without loss of generality, we assume the attacker is player 2 and the corresponding useful observations are message 1.1 and 1.2. The same lines of reasoning also applies to the case where the attacker is player 1. Also, all *size*s mentioned in this proof are the size of the application-level plain-text HTTP payload. We do not take into account any size variations like paddings and chunkings introduced by the underlying TLS protocols, and this is OK because if we prove that the program is not vulnerable for the sophisticated attackers to whom TLS is not a problem, it must also not vulnerable for those who are less sophisticated. Message 1.1 and 1.2 are json payloads containing the following fields: 1. opponentCards; 2. tableCards; 3. serverMessageType; 4. youAre; 5. holeCards; 6. rendering; 7. Padding. Field 1 to 4 are always of fixed size for a given message. Sizes of field 5 and 6 only depends on the secret (i.e. the two hole cards held by the victim), and field 7 has a randomly generated length from 5KB to 95KB. If we use $(s1;\ s2)$ to denote the victim's two secret hole cards, $(o1;\ o2)$ to denote the size of message 1.1, 1.2 respectively, C1 and C2 to denote the total size of 1 to 4 for message 1.1, 1.2 respectively, $F1(x)$ and $F2(x;\ y)$ to denote the rendering size of card x and card pair $(x;\ y)$ respectively, and $rand(X;\ Y)$ to denote a random number uniformly distributed in range $[X, Y)$, then we have the following equation:

$$o1 = F1(c1) + C1 + rand(5000;\ 95000)$$
$$o2 = F2(c1;\ c2) + C2 + rand(5000;\ 95000)$$

Equation A-7: Poker Challenge Message Size Equation

The attacker's job then is to infer the value of $c1$ and $c2$ given the pair $(o1;\ o2)$. The budget given in this problem is 175, and 2 units of cost must be used to obtain $o1$ and $o2$. That leaves 173 units of cost to the attacker. As the question only asks for the card pair regardless of the order, guessing $(x;\ y)$ is essentially the same as making two guesses $c1 = x;\ c2 = y\ and\ c1 = y;\ c2 = x$ from the attacker's perspective. We therefore double the budget of the attacker from 173 to 346. Since the attacker can construct the mapping $F1$ and $F2$ offline, the only unknown term in the above equation is the random term. Each secret pair $(c1;\ c2)$ may correspond to any observation of the form $(F1(c1) + C1 + x;\ F2(c1;\ c2) + C2 + y)$ where x and y are in the range 5K and 95K. If we visualize the secret and the observation in a 2D plane, each secret can be represented by a single 2-D point and the corresponding observation space can be represented by a 90K * 90K square with $(F1(c1) + C1 + 5000;\ F2(c1;\ c2) + C2 + 5000)$ act as one of the vertices of the square.

If the required success rate is 100%, we can simply prove the infeasibility of the attack by finding an area where more than 346 such squares overlap. Proving the success rate is no larger than 95%, on the other hand, requires more advanced reasoning. The basic idea is to again find an area A where B squares overlap (B > 346). In areas like A, the attacker is guaranteed not to succeed with 100% rate. Then we show that the area A is so large that it show up frequently enough to inevitable lower the overall success rate below 95%. First, we claim that if the observation $(o1;\ o2)$ falls into A, then the optimal strategy the attacker can use here is to guess 346 of the B corresponding secrets uniformly at random. This is due to the perfect symmetric nature of this problem: the square that corresponds to each secret has the same size and shape for all secrets, so any bias the attacker put on any secret will inevitably result in a suboptimal strategy for some other secret. Now suppose the area A encompass P observations. We know that for any given secret, there are in total $90K\ x\ 90K\ =\ 8100M$ possible observations and each observation has the same probability to occur. So the probability of the observation that falls into area A is $\frac{P}{8100M}$. For these observations, the optimal strategy has a success rate of $\frac{346}{B}$. Even if we assume that for any other $(8100M\ -\ P)$ observations the attacker is able to reach a success rate of 100% (which is of course merely an upper bound), the total success rate R for this particular key would be:

$$R = \frac{P}{8100 * 10^6} * \frac{346}{B} + (1 - \frac{P}{8100 * 10^6})$$

Equation A-8: Poke Challenge Success Rate Equation

So as long as we can find one area A with $R\ <\ 95\%$, we know that there exist at least one key for which the attacker cannot infer with 95% success rate. According to the STAC operational definition, this proves that the program is not vulnerable. In the proof 009 subdirectory there is a C++ program that tries to read the problem configuration and find a rectangular area A with $R\ <\ 95\%$. It also takes into account which player theattacker is as well as what skin is used for rendering: different player/skin combination may lead to different $F1, F2, C1\ and\ C2$. Our program manages to find A for every combination."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Our initial analysis depends on the assumption that the implementation of ThreadLocal.nextInt() produces a perfect uniform random distribution. However, we are informed by the Vanderbilt and GrammaTech team that this function is not cryptographically secure: the entire random sequence generated by ThreadLocal.nextInt() is completely determined by the initial seed. This means that if the random seed is known to the attacker, the size of the paddings can be easily inferred. We added another program proof2.cpp in the proof 009 subdirectory to show that the secret information can be revealed with no more than 2 oracle queries once the random padding size is known to the attacker. The random seed is generated per thread, and the server serves each client with a separated thread. Therefore, only the sizes of the

messages that are sent to the opponents are useful to infer the random seed used by the victim thread. Among the 6 messages mentioned in our original answer, one can only calculate the padding size for one message (i.e. the first one). From the discussion it is clear that calculating the random seed from only one observation is too prohibitive to be feasible on the reference platform. From the practical point of view, the random distribution assumption still holds to some extent. In summary, despite the fact that we learn the existence of an additional attack surface during the live collaboration, we do not see strong evidence that exploits can be crafted for it within the budget."

Live-Collaborative Evaluation:

Through collaboration Colorado corrected a statement from their take-home analysis that the PRNG is secure. Working with collaborators, the team observed that while determining the PRNG seed would enable a side channel, the fact that each thread has a different seed coupled with the question's restriction of the attacker to a single game with the victim user prevents a vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.8.2 Question 031 (SC Time, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"No, the poker application does not have a side channel vulnerability in time. We investigated the application and found that the application does leak side channel information in time. However, the variance in the information leaked does not allow the attacker to gain any substantial knowledge about the cards held by the opponent. The attacker is unable to gain enough information to have a 95% probability of success given the number of operations allowed. The poker application leaks information about the cards in the response times from the server to the player. During game status updates, the board is rendered server-side by the function com.ainfosec.STACPoker.PokerServer.sendTableToClient for each player, and sent to them over the network. Each card is drawn separately on the table, and an individual table is draw for each player. If the attacker connects to the server first, they can control when the server will deal each of the four hole cards, as the server waits until they send an acknowledge message to deal the next card. By measuring the time between sending the acknowledgement message and receiving the next game status update, the attacker can estimate how long it took for the server to render the table. The time taken is a combination of server time and network time. For this side channel to be useful, the noise in the timing should be small relative to the expected time so that accurate information can be deduced from the card. The attacker can control when the server deals the next card while dealing of the hole cards and can measure get an approximate time to render the board. The attacker can then infer the time taken for dealing one card by subtracting the timings for rendering the board with or without that card, e.g. the time inferred for the first opponent hole card is the difference between the response time for the board with the two attacker cards and the response time for […] the board with the two attacker cards and a single card for the opponent. The response time includes drawing the board for both attacker and opponent. These timings can give the attacker an estimate of time for rendering each of the two cards that the opponent has. If these timings depend on the card, they may allow the attacker to infer information about the cards. However, being able to infer information depends on how much variation there is in the measurements versus how different the timings are for the different cards. For each game these cards are random and dealt only once, and therefore the timings for

individual cards can only be measured once. The precision of this information cannot be increased by taking additional measurements. The timings were measured over 50000 runs of the poker application. In each run the time was measured by the attacker between sending an acknowledge message and the server's response with a game status message with a new rendered board. The estimate times per card were computed by taking the difference between these timings. We did not see much information gain from requesting a specific skin, but the attacker requested the yellow skin because preliminary experiments showed the lowest variability with the yellow skin. The suit did not have an impact on the timings for either card dealt, as show in figure 26, and no information could be derived from it. The number of the card did have an impact on the time to render. More complicated cards, increasing numbers and face cards, had proportionally longer timings. The timing distribution of the cards have clearly distinct peaks, and the distributions of the timings can be distinguished. This is seen in figure 27. However, the distributions of the timings have significant overlap, and is not clear which distribution any given timing can come from. Figure 28 shows the 5-th and 95-th percentiles for […] timings for each card number. A large majority of timings could have come from any of the numbers. The worst-case timing for identifying the card based on timing would be if the observed card is near the mean observed timing. The standard deviation observed across the network for the timing for a specific card is 15.46 ms. The high variance in timings means that the timings for different cards overlap. Figure 29 shows that most of the card combinations are close to the mean. […]”

Take-Home Evaluation:

The challenge program leaks whether a new skin has already been rendered for a given card. This side channel can be used by the attacker to segment the cards into two groups. Oracle queries can be used to resolve the residual entropy. Colorado correctly concluded that there is not a vulnerability that allows for individual cards to be distinguished.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.1.2.9 Searchable Blog

A.5.1.2.9.1 Question 027 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Yes, there exists an algorithmic complexity vulnerability in time. When the user submits a blog for ranking, the synchronized procedure performs Ranking (RankingService) will be called and this method will block requests from other benign users. Since all the submitted blogs will be written to the same file called \userblog.html" on the server side, we just need to upload a malicious blog that contains a hyperlink that points to:

\userblog.html”, <html><body><a href="/blog/userblog.html"/></body></html>

and repeat the same request multiple times (i.e. 2 times). For each request, the blog uploaded by the attacker will be converted into a vector of the form [0; 0; ::: ; 0; 1]. This vector will cause the

result of the matrix multiplication (M fi b) to be very small. As a result, the fixed-point computation in the while-loop will converge very slowly. Figure 31 shows the values of vector b in the last few iterations. Eventually, the attacker's input will cause procedure estimateStationaryVector to run for about 10 secs with > 10K iterations. With the given budget, we were able to block the requests from benign users for 10 seconds. […]"

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.10 SimpleVote 1

A.5.1.2.10.1 Question 019 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Yes, there exists an algorithmic complexity vulnerability. The part of the procedure com.cyberpointllc.stac.algorithm.funkyFunction shown below shows a loop in which a BigDecimal number is updated by multiplying it with another BigDecimal number. Such a multiplication can double the precision and thereby roughly doubles the runtime and used memory in each update step. Hence the loop has a worst-case exponential complexity and the user can control the loop length (parameter n). […] Although parameter n is user-controlled it is restricted: The call to funkyFunction is sanitized by the method isInBounds, which limits n to be at most MAX N = 995 initially. Moreover, the challenge application consists of a table that prevents k (the start of the loop) to be at most n - 6. However, it is possible to increase MAX N by 7 by calling isInBounds with MAX N although it returns false and therefore blocks calling funkyFunction. Increasing MAX N also breaks the invariant that the maximal difference between n and k is at most 6. In order to reach the runtime of 300s, one only requires to increase MAX N three times and afterwards call funkyFunction with n = 995+2fi7 = 1009. The highest entry in the table is 988, so this will be the value of k. Hence the loop will be performed 21 times. The runtime will exceed 300seconds as the loop is exponential."

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.10.2 Question 025 (SC Time/Space, Unintended Not Vuln., Take-home: Yes, Live: No)

Take-Home Response:

"Yes, there exists a side-channel in space and time from which an attacker can obtain any single ballot. The vulnerability is possible because any user that can login the system can access any election by simply visiting the URL https://serverNUC:8443/ballots/XXX with XXX being the digits of the selected election. After logging into the system, the source code of the overview page shows the digits for each election. After accessing the election page via the above-mentioned URL, the user is asked to enter a registration key, which is unique for each ballot. Entering the registration key will allow the attacker to access the ballot (without requiring any login details of the target voter). The attack consists of two parts: 1) obtaining the registration key of the target user using aside-channel in time; and 2) determining whether the target voter accessed a ballot using aside-channel in space. Regarding the side-channel in time: The registration key consists of two parts, each of 6 characters. Let's call them the matching part and the index part. One can permute the characters in such a way that the first 6 characters belong to the matching part and the second 6 characters belong to the index part. This is also what happens in the application when the registration key is validated. The runtime of the validation depends strongly on the index part of the registration key, see Figure 37. The larger the index part, the higher the validation time. The difference between two consecutive keys, the runtime difference is roughly 0.1 seconds, which makes it easy to pinpoint the key. […]"

Take-Home Evaluation:

The challenge program contained an intended vulnerability whereby a registration key is transformed to an integer in a deterministic manner, and the runtime for processing a registration key is linearly correlated to the value of the key. The side channel strength is dependent on the secret range, and the application code must unambiguously provide this. It was assumed that there was a 1-to-1 mapping between a registration key and a translated integer value of the key. However, in the application code 3 specific characters in the registration key are replaced with 3 specific numbers during the transformation from registration key input to an integer. The challenge program therefore allows for 2^6 possible registration keys to map to the same integer. There are not enough oracle queries to resolve this. This unintended not-vulnerable case was identified by Vanderbilt.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After discussions with the Vanderbilt team, we have concluded that the exploit described below does not achieve the desired result within the budget, we justify our change below. Based on discussions with the Northeastern team, we now believe there is no side-channel in time that can be exploited within the budget. Our original response is not valid because we assumed that there was a 1-to-1 relationship between kstr and nstr in the method PermissionAuthenticator.confirm. The Vanderbilt team pointed out that there is in fact a many-to-1 relationship between the kstr and nstr, caused by the replacement of characters to digits, A to 7, C to 6, E to 3 (see control flow graph in Fig 39), which maps, e.g., originally distinct keys with an A and a 3 in the same position to the same string, which is irreversible, as information is destroyed. This conflation of characters can only happen at the lower 6 characters of the string, since the characters 3, 6, and 7

are disallowed in the upper 6 characters of the original string. Thus, for every unique value of nstr, there are $2^6 = 64$ possibilities for the kstr. With the perfect observations from the timing side channel, we can still compute a matching string and pass the confirmEssence test using funkyFunction. However, we cannot, within 25 operations, recover the original key, which is necessary to pass the fetchDirectVoterService().confirmPermissionKey() in PermissionAssesGuide.handlePost."

Live-Collaborative Evaluation:

Colorado correctly determined this is not vulnerable through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.11 SimpleVote 2

A.5.1.2.11.1 Question 008 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Yes, there is an algorithmic complexity vulnerability in space for simplevote 2. The vulnerability happens when two servers try to sync. The memory usage will get blown up to beyond 1GB for each server. The exploit is under the exploit directory simplevote 2/exploit 008.There are two ways to trigger servers to sync. One is to simply wait for 1 hour after starting two servers, because there is a timer which will trigger server sync every 1 hour (incom.cyberpointllc.stac.basicvote.compilation.AccumulationChore.isDueForUpdate). This will eventually invoke c.c.s.EasySelectionServiceImpl.updateReports. The other way is to submit a vote 10 times. Every hit to the "vote" button on webpage"/ballots/401" will invoke c.c.s.basicvote.handler.EditRecordManager.handlePost, which would invoke c.c.s.basicvote.EasySelectionServiceImpl.addOrUpdateRecord, which would finally invoke c.c.s.basicvote.EasySelectionServiceImpl.updateContest and count the vote as one update to its corresponding contest. Every 10 updates to a contest will also invokec.c.s.basicvote.EasySelectionServiceImpl.updateReports. However, if a vote is valid, the voter does not have a chance to vote for 10 times for a single ballot. The only way for a voter to vote for 10 times is to submit an invalid vote each time. The counter would still be increased even when the vote is invalid, and when the counter reaches 10, it will trigger server sync. […]"

Take-Home Evaluation:

Colorado identified an unintended vulnerability similar to the one reported by Two Six Labs.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.12 STAC SQL

A.5.1.2.12.1 Question 013 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Yes, there is an algorithmic complexity vulnerability in space that causes the challenge program to store a file with a logical size that exceeds 2 MB. The attack, demonstrated by the scripts in folder exploit 13, exploits a bug in the Huffman encoding2 module for the Blob data-type. Specifically, whenever a field of type Blob is being updated and the new value contains a symbol that does not appear in the current value, the application calls methodcom.stac.encoder.HuffmanCoder.updateTree which can create an extremely inefficient encoding for certain symbols. The inefficiency stems from the fact that updateTree always grows the height of the Huffman tree whenever a new symbol appears in the new value.”

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“The logic behind our attack remains the same. The only improvement is in the exploit to make the attack within budget. The modified script (inside folder exploit 13) sends all the commands at once in the server, avoiding this way the SSL overhead imposed by our previous script.”

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.13 Stegosaurus

A.5.1.2.13.1 Question 003 (SC Time, Unintended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“No, there is no side-channel vulnerability in time from which the attacker can obtain the message given the input budget. To obtain the message, there are two possible modes of attack: either obtain the 4096-bit secret message directly from the timing side channel or obtain the 128-bit secret key from the timing side channel and then obtain the message from the decoder by providing it with the secret key. The block diagram for the encryption and decryption operations are shown in Figure 43 and Figure 44. From the decryption process it can be seen that the message can be derived from the decryption operation if the secret key is known. We assume the encrypted image is also known to the attacker since the server does not properly authenticate its users and therefore it is easy for the attacker to retrieve the image of a user after that user finishes uploading. Initially, we assume that inferring the secret key is the easier option. This assumption is justified later using our tools. The timing side channel is the time it takes for the victim to encode the message into an image using the secret key. This encoding operation takes place incom.bbn.Stegger.hide(). This method has three nested loops: the outermost for loop iterates over each character in the message, the for loop nested inside this loop iterates over each of the 8-bits in the binary representation of each character, and finally there is a while loop nested inside both the for loops. Since the attacker must deduce which among the 2128 possible secret keys was chosen by the victim, the combination of these three loops must make different number of iterations for each possible secret key (since the two for loops are not affected by the […]First, we fix both the message and image, and we randomly generate the secret key each with different values and different number of 1's and 0's in the key. The result of this experiment for 10,000

secret keys is shown in Figure 45, where we apply our tool Discriminer to cluster the possible observations of this process. This experiment clearly shows that we can only observe three timing classes (clusters) when randomly generating keys. In the second experiment, we justify our earlier assumption that attempting to directly deduce the 4096-bit message instead obtaining it indirectly by deducing the 128-bit secret key from the timing side channel is not the better option. We consider the execution time as a function of different messages. For this experiment, we consider 200 secret keys where each secret key runs over 50 randomly generated input messages of lengths between 2 and 800 bits (hence we have 200 different functions, one for each secret key). Note that we fix the image size for all experiments. Overall, we collect the information about how 200 timing functions vary over the messages shown in Figure 46. (Figure 47 is a\stripped down" plot showing 4 functions out of 200 from Figure 46). To find the number of clusters in execution time, we applied our tool SCHMIT, which finds only 8 clusters or distinguishable timing observations from overall 200 secret keys. […]”

Take-Home Evaluation:

The challenge contained an intended vulnerability due to a race condition. An attacker can use this to leak the secret bit-by-bit. However; the intended vulnerability was determined to be insufficiently strong.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.14 StuffTracker

A.5.1.2.14.1 Question 030 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Yes, there is an algorithmic complexity vulnerability in space as demonstrated by the script in folder exploit 30. This exploit is identical to the Billion laughs attack. The application accepts XML format as input without performing any sanitization when a user uploads a product.”

Take-Home Evaluation:

Colorado has identified the same out-of-scope billion lols-style vulnerability as other teams.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“During the collaborative engagement, we were pointed to another vulnerability that triggers core methods of the application and not only the XML parsing library. We retrieved the malicious input (new-input.xml inside directory exploit 30) from our collaborators, and we were able to reproduce the vulnerability. This attack starts with a "/stuff" POST which modifies the store variable in StuffTrackerService. As a result, the application stores some large les inside the data directory. Then, the exploit uses a "/stuff?term=sta" GET to call store.search, which in turn

calls DocIndexer.search. In DocIndexer.search, DocIndexer.readResult is called for each entry in SDLDataStore.indexes, and this will cause a lot of data to be read from the aforementioned les into memory causing the application to exceed the specified resource usage limit."

Live-Collaborative Evaluation:

Addendum to take-home evaluation: during collaboration Colorado was convinced of an additional unintended vulnerability reported by Draper. Following the take-home engagement, Draper's vulnerability was evaluated on the reference platform and found to be non-vulnerable. While this does not impact the post-engagement evaluation, further investigation is required on this second potential unintended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.1.2.15 Tawa FS

A.5.1.2.15.1 Question 012 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Yes, there exists an algorithmic complexity vulnerability in time. The recursive procedure com.bbn.Filesystem.findAllFiles (shown below) can result in an infinite loop, because it is allowed to create directories in which the names can include the symbol /, such as directory name a/a. The control ow graph (figure 51) and the call graph (figure 52) for com.bbn.Filesystem.findAllFiles show the recursive loop. […]"

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2 Draper Labs

A.5.2.1 Draper Labs Overview

Draper answered 21 questions during the take-home engagement with a 24% accuracy. During the live engagement the team had the highest increase in accuracy (57 percentage-points) to 81%. Draper labs had less than 50% accuracy on all question categories during the take-home engagement. The team had the lowest accuracy in both the take-home and live engagement. In cases where the team changed their answer from the take-home to the live engagement, they had a 100% accuracy.

During the take-home engagement, on BattleBoats 2 Question 021, the team's response indicated that to exceed the resource usage limit, the memory consumption of both the attacking peer application and the peer under attack were added. If this is accurate, then it is against the operational definition for measuring resource consumption as the consumption of only the peer

under attack is to be measured. In BattleBoats 2 Question 21 and Searchable Blog Question 27, the tool output in the response was the intended vulnerability region (*HitLocator.computeHitTime()* in BattleBoats 2, and *estimateStationaryVector()* in Searchable Blog); in both cases the team declared that region of the application non-vulnerable.

A.5.2.1.1 Take-Home Engagement Scores

Table A-85: Engagement 5 Draper Labs Take Home Engagement Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	5	2	40	2	40
SC Time	5	2	40	1	20
SC Space/ Time	1	0	0	0	0
AC in Space	5	3	60	1	20
AC in Time	5	1	20	1	20
Total	**21**	**8**	**38**	**5**	**24**

Table A-86: Engagement 5 Draper Labs Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	16	6	**38**	3	**19**
Not Vulnerable	5	2	**40**	2	**40**
Yes Answer	9	6	**67**	3	**33**
No Answer	12	2	**17**	2	**17**

A.5.2.1.2 Live Engagement Scores

Table A-87: Engagement 5 Draper Labs Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	5	4	80
SC Time	5	4	80
SC Space/ Time	1	1	100
AC in Space	5	4	80
AC in Time	5	4	80
Total	**21**	**17**	**81**

Table A-88: Engagement 5 Draper Labs Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	16	13	**81**
Not Vulnerable	5	4	**90**
Yes Answer	15	13	**87**
No Answer	6	4	**67**

A.5.2.2 Draper Labs Responses

A.5.2.2.1 Accounting Wizard

A.5.2.2.1.1 Question 011 (SC Time, Unintended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"This vulnerability is a side channel vulnerability in time from which an attacker with Employee credentials can discover the project expenditure limit for any project that they work on. Script expenditurelimit.py exploits this vulnerability. Script expenditurelimit.py works by charging hours to a project until the expenditure limit is reached. Script expenditurelimit.py should be executed after logging in the employee. An employee attempting to charge more hours than the current remaining budget on a project will cause the server to respond faster than if the budget can afford that charge. A file will be updated with the update remaining project budget in the

case that hours are charged. No file is updated if trying to charge more hours than the budget allows. This difference causes the discrepancy in the response times between the two cases.

The maximum expenditure limit for any project is 3000000. Attempting to create any project with a larger expenditure limit will result in a JSON response with an error message such as the following: {"code":500,"message":"Project required expenditures [NEWPROJECT] have exceeded the maximum allowed hours [{1}]"}

Also, the cost of an hour for each employee is 100. (Based on field com.bbn.accounting.objects.FileStore.costOfHours). Therefore, the number of hours an employee can charge for any project is 3000000/100 = 30000. Finally, script expenditurelimit.py works by trying to charge a number of hours stored in variable chargeIncrement. chargeIncrement is initialized to 30000 (max # of hours that can be charged). Variable currentHoursCharged stores current amount hours charged. currentHoursCharged is initialized to the current amount of hours charged on the accessed using a HTTPS GET request (i.e. https://serverNuc:4567/employee/getHours/). chargeIncrement is reduced by half if chargeIncrement is greater than the remaining project budget. Otherwise, currentHoursCharged is incremented by chargeIncrement. This loop continues no more hours can be charged (i.e. charging a single hour fails). After exiting the loop, the expenditure limit is equal to 100*currentHoursCharged. Vulnerable method(s): com.bbn.accounting.security.EmployeeManager.updateHours”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The reported side channel appears similar to the one reported by Northeastern, using employee hours rather than expensed items. This relies on an underlying assumption that an employee can set the number of hours on a project to 0. While this is possible for the employee’s own hours we (the EL) don’t believe this is possible for other employee’s hours or charged items.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“[…] Our exploit script took advantage of the following side channel in method com.bbn.accounting.security.EmployeeManager.updateHours, conditional branch if (!FileStore.hoursFit(project, e, employeeHours)) An employee attempting to charge more hours than the current remaining budget on a project will cause the server to respond faster than if the budget can afford that charge. A file will be updated (by method FileStore.writeFileStore) with the update remaining project budget in the case that hours are charged. No file is updated if trying to charge more hours than the budget allows. This difference causes the discrepancy in the response times between the two cases. […]”

Live-Collaborative Evaluation:

There was a mistake in packaging this challenge program: the application came provided with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects’ contents or states.

The unintended side-channel vulnerability identified by Draper and several STAC teams was confirmed.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Potentially vulnerable methods which are not problematic:
* com.bbn.accounting.objects.FileStore.readFileStore
* com.bbn.accounting.objects.BaseObject.write
* com.bbn.accounting.objects.FileStore.newBudget
* com.bbn.accounting.objects.FileStore.newItem
* com.bbn.accounting.objects.FileStore.newProject
* com.bbn.accounting.objects.FileStore.newHours
* com.bbn.annotations.processors.FileStoreProcessor.process
* com.bbn.accounting.budgeting.BudgetSolver.getAffordableLines
* com.bbn.accounting.tools.database.GroupSumMemoizer.<init>
* com.bbn.accounting.budgeting.BudgetSolver.prepareSetSpecifier
* com.bbn.annotations.processors.UtilityPackageProcessor.process

Potentially vulnerable methods which have (sufficiently!) sanitized

* com.bbn.accounting.objects.BaseObjectMarshaller.marshallAnnotatedFields
* com.bbn.accounting.logging.Logger.supportsTag"

Take-Home Evaluation:

The challenge program contains an intended vulnerability. An attacker can trigger verbose logging by causing the challenge program to throw a custom exception.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"If the locale does not exist, the logging level is set to trace. When trace logging is enabled, method com.bbn.accounting.logging.Logger.trace() will write messages to a file. Furthermore, when trace logging is enabled, the children ids of a project will be logged allowing at least 1MB to be written to disk. Specifically, the size of directory accounting_wizard/challenge_program/.store can become greater than 1MB. […]"

Live-Collaborative Evaluation:

Draper identified the intended vulnerability through collaboration

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.1.3 Question 023 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Potentially vulnerable methods which are not problematic:

*com.bbn.accounting.objects.BaseObjectMarshaller.marshallAnnotatedFields(Lcom/bbn/accounting/objects/BaseObject;)[[B

*com.bbn.annotations.processors.FileStoreProcessor.process(Ljava/util/Set;Ljavax/annotation/processing/RoundEnvironment;)Z

*com.bbn.accounting.budgeting.BudgetSolver.getAffordableLines(Ljava/util/ArrayList;J)Ljava/util/Collection;

*com.bbn.accounting.tools.database.GroupSumMemoizer.<init>(Ljava/util/ArrayList;)V

*com.bbn.accounting.budgeting.BudgetSolver.prepareSetSpecifier(Ljava/util/ArrayList;J[Z)J

Potentially vulnerable methods which have (sufficiently!) sanitized

*com.bbn.accounting.objects.BaseObjectMarshaller.marshallAnnotatedFields"

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The complexity of the budget settling algorithm can be used to cause the benign user's response time to exceed the resource usage limit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Method com.bbn.accounting.budgeting.BudgetSolver.getAffordableLines has a AC time vulnerability. It is called by method getAllowedExpenditures also defined in the same class (BudgetSolver). Method getAllowedExpenditures is called when an employee requests to update his or her hours to check that the cost of the additional hours fit within a project's budget. Method getAffordableLines has exponential runtime in the worst case with respect to the number of items in the project. Method getAffordableLines will iterate through each subset of items of the project to find a subset of items that fits with in the budget when new hours are charged."

Live-Collaborative Evaluation:

Draper identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"The server's response messages sizes are constant when ordering items and when querying budget information."

Take-Home Evaluation:

There is a side channel in the file sizes stored to disk. A user request can be used to retrieve this data. Draper's response indicated analysis of server response sizes. As there are no passive operations available, this avenue of attack should have been eliminated.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"When adding an item to a project, the project file size increases by 44 bytes. […] Finally, the number of budget items is equal to the following $\frac{(project\ file\ size)-24-stringLength(project\ name)}{44}$„

Live-Collaborative Evaluation:

Draper identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.2 BattleBoats 1

A.5.2.2.2.1 Question 006 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“Tool output files:

- battleboats_1_aprove_results.txt

Potentially vulnerable methods which are not problematic:

Potentially vulnerable methods which are not reachable:

Potentially vulnerable methods which have (sufficiently!) sanitized input:

Other remarks:

Since the commands that are available to the player are dynamic we suspected that it could be possible to somehow add an infinite amount of available commands. However, we did not find a way to add a duplicate instance of a command to the list of available commands. This would be necessary in order to increase the runtime of the 'help' command.”

Take-Home Evaluation:

The challenge contains a vulnerability. An attacker can submit an input that causes the projectile to never reach the ground. As this is a terminating condition, a benign user request sent after this attack would be delayed accordingly. The goal is not necessarily to cause the ‘help’ command to exceed the resource usage limit. This is simply a representative user command. The user has an application that accepts remote commands the attacker is then free to use whatever sequence of inputs is accepted by the remote endpoint to impact the benign user. It is unclear from the response if this objective is clear.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“If a large (positive or negative) value is passed to the vulnerable method, it is executed for a long time. The cause of this behavior are two loops in the method body, which ensure that the degree value is positive and less than 360, before converting to radians. The loops add and subtract 360 to the value respectively, until it is in range. The vulnerability can be triggered by the attacker during a game using the following example command: shoot 0 90 10e200
Note that it does not matter which of the two angles we choose to be the large value, as they both get converted to radians with StrikeLocator.toRadians(double degree).”

Live-Collaborative Evaluation:

Draper identified an unintended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.2.2 Question 028 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“Tool output files:

- [battleboats_1_aprove_results.txt] (battleboats_1_aprove_results.txt)

Potentially vulnerable methods which are not problematic:

Potentially vulnerable methods which are not reachable:

Potentially vulnerable methods which have (sufficiently!) sanitized input:

Other remarks:

If the board size is 10x10 the oracle can be used to determine the position of the cannon, however that is not a universal approach. We do not see a possibility for an attack, since we are not allowed to monitor a whole session, but only active requests and the oracle.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.3 BattleBoats 2

A.5.2.2.3.1 Question 010 (SC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“We suspected that a difference in response time can be measured between hits and misses, i.e. shots that hit the board and shots that miss the board. In class `OceanPanel` the method `pullOceanSquares` exits early on a miss but calls the methods `takeHits` and `updateHitsAsSunk` on a hit. However, in an experiment we could not detect a measurable difference in response time. We tried this approach since in contrast to the related question 21 for program `battleboats_1` we are allowed to observe a whole session for very little cost.

If the board size is 10x10 the oracle can be used to determine the position of the cannon, however that is not a universal approach. We thought about using the oracle in combination with the approach described above but could also not find a vulnerability.”

Take-Home Evaluation:

The challenge contains an intended vulnerability in detecting shots that hit the board vs shots that go off the board. Draper’s response states they suspected a difference in response time between shots that hit the board vs miss the board. Draper adds that they were not able to detect this timing difference. The side channel is present; therefore, it is unclear why Draper reported no difference in response times. Perhaps this was not done on the reference platform/reference network environment.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“A difference in response time can be measured between hits and misses, i.e. shots that hit the board and shots that miss the board. In class OceanPanel the method pullOceanSquares exits early on a miss but calls the methods takeHits and updateHitsAsSunk on a hit. […]”

Live-Collaborative Evaluation:

Draper identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.3.2 Question 021 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Potentially vulnerable methods which have (sufficiently!) sanitized input:

- HitLocator#computeHitTime(): The bound on the number of iterations limits to runtime to constant.

Other remarks:

When starting a game using the `start.sh` and `bening_basic.sh` scripts, which in turn use the provided `player1.expect` and `player2.expect` scripts, we already reach space consumption of about ~400M. With a small change to `player2.expect` we get higher memory usage, and occasionally a crash of the "client" (modified player 2). The change involves player 2 making nonsensical moves close to the end of the game (where the unmodified `player2.expect` wins) and not waiting for expected output to the terminal. The memory usage was observed using `pmap -x` on the two `java` processes.

6512 total kB	15801608	380624	363068
6587 total kB	15868172	664908	647388

The messages from player 2 to player 1 are small (around ~100B on average, measured using Wireshark) and thus we conclude that a vulnerability is present and can be exploited using the given budget.”

Take-Home Evaluation:

The challenge contains an intended vulnerability. The algorithm to calculate hit time for a narrow band of inputs can be made to exceed the resource usage limit. From Draper’s response it is not clear than the vulnerability identified satisfies the requirements of the reference platform + operation definition. Both the attacker and benign applications appear to be run simultaneously on the same device and Draper appears to have reported the combined resource usage of both applications. Additionally, in Draper’s response the method containing the actual vulnerability is reported as having a constant runtime bound which is not accurate.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“The attacker can cause high memory usage in the benign player's client by issuing a shoot command with a very small height value and a small velocity value, like the following. […] In the benign player's client the values will subsequently be fed into a HitLocator instance as BigDecimals. When the computeHitTime method is called, the computation causes allocation of

large BigDecimal instances as well as a large HashMap instance storing intermediate values of the computation. When many shoot commands like above are issued the garbage collector is not able to free the HashMap instance in time and the client memory exceeds the limit of 800 MB."

Live-Collaborative Evaluation:

Draper identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.4 IBASys

A.5.2.2.4.1 Question 015 (SC Space, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"If the client sends an invalid image, then the server returns a fresh AES key (generated via methods from the java standard library) instead of the valid token. This AES key is in no way related to the passphrase-image or the token of the target user. The same holds for the size of the server's reply."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attacker can piggyback a malformed request off an invalid but well-formed request. The malformed request can take advantage of a space SC to leak the passcode string used to validate a user image. This leaked string can then be used to generate a passcode image that will be accepted by the application.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.2.2.4.2 Question 024 (SC Time, Intended Not Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Input files required for ~~(infeasible)~~ vulnerability:

[onebit.jpg](http://aprove.informatik.rwth-aachen.de/downloads/onebit.jpg)

[zerobit.jpg](http://aprove.informatik.rwth-aachen.de/downloads/zerobit.jpg)

[IbasysSideChannelSrc.zip](http://aprove.informatik.rwth-aachen.de/downloads/IbasysSideChannelSrc.zip)

[IbasysSideChannel.jar](http://aprove.informatik.rwth-aachen.de/downloads/IbasysSideChannel.jar)

High level description of vulnerability:

Only 32 bits of the password image are actually checked for authentication. The reason is that the image is first cropped to 100x100 pixels in `ScalrApplyTest.testApply1`. Then, the RGB

values of these 100x100 pixels are store in an array. Then for each 78-th entry of this array, the information whether it is even or odd is computed, resulting in 128 bits. Every 4-th of these bits is compared with the polarity of the corresponding entry in `users.data`, i.e., just 32 bits need to be known in order to pass the authentication.

The loop comparing these bits has an early-exit condition. Given the input budget of 3500 operations, we have more than 100 operations to recover a single bit, i.e., we can issue 50 requests where the first bit is 1 and 50 requests where the first bit is 0 and compare the average runtime.

However, it is unclear whether the difference in runtime is indeed measurable and implementing such an attack for testing is quite complex.

Potentially vulnerable method(s):

`ImageMatcherWorker.run`

Additional notes:

I tried to implement the attack by creating two pictures which are almost equal but differ in the first relevant bit. Then, I adapted the client in order to measure the time between sending all data and receiving the response as precisely as possible. However, there's no significant difference. Thus, I think the attack is not feasible. After a bit more fine-tuning, I can measure a tiny difference in timing. After closing all other applications, `IbasysSideChannel.jar` reliably found out that the first relevant bit for the authentication of "hansolo" is `1` 10 times in a row. However, the attack is fragile. E.g., on my laptop it failed when I ran it from eclipse (instead of running the jar) and it also failed when I moved the terminal from one screen to another one while the jar was running, so keep your hands away from the keyboard. To double-check the results, I flipped the first bit in the user database. Thus, the attack should now yield `0` most of the time. Unfortunately, it doesn't. Thus, as it is the attack still does not work. However, I still tend to answer "yes", as implementing the attack might just require further effort. In especially, there's no good argument why just checking 32 out of ~280k bits of the passcode-image and doing so in a loop with an early exit condition should be safe. To reproduce the (failing) attack, download `IbasysSideChannel.jar` and put it in the same folder as `public.der`, start the server **with -Xint to disable just in time compilation**, close all other applications, invoke the jar with `zerobit.jpg` as first argument, `onebit.jpg` as second argument, and the number of requests which should be sent to the server for each image as third argument, and keep your hands away from the keyboard. After a while, you will get the median of the runtime measured for the zerobit-requests and the onebit-requests."

Take-Home Evaluation:

The challenge does not contain an intended SC Time vulnerability. The potential vulnerability reported by Draper is not believed to be sufficiently strong to satisfy the operational budget. The loop identified by Draper in ImageMatcherWorker.run() is one that was flagged prior to the engagement as a potential red herring for an SC Time; however this looping structure is actually used in the intended SC Space vulnerability in Question 015.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.2.2.4.3 Question 029 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The method `IBASysServer.init` logs an exception to a file if a session is established successfully, but parsing the payload fails (in `LoginManager.parsePackagePart`). The log-entry contains `session.databuf`, which is initialized to an array containing ~1000 zeros at the very beginning of `parsePackagePart`. The resulting log file has size ~167kb. To establish a session successfully and trigger the logging, a message of size 272b has to be sent to the server. Thus, the input budget of 1mb allows for more than 3500 messages. However, ~1000 messages are already sufficient to exceed the ressource limit of 150mb."

Take-Home Evaluation:

Draper identified the intended vulnerability in the application.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.5 Medpedia

A.5.2.2.5.1 Question 009 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Potentially vulnerable methods which are not problematic:

ContentController.getMasterPage()

The output on the response buffer depends (of course) on the size of the output, but is padded by a random string of length random.nextInt(2016) + 32.

The ability to predict the size of this padding is tied to the ability to predict this random number.

The random number generator is initialized by a call to ContentController.getRandomBytes(20). This seems reasonably secure as it is read from /dev/urandom, stripping away 0 bytes. The random number generator itself seems reasonably secure, though hard to test.

Potentially vulnerable methods which are not reachable: N/A

Potentially vulnerable methods which have (sufficiently!) sanitized input:

Analysis particularly tricky because of heavy use of reflection."

Take-Home Evaluation:

The challenge contains an intended SC Space vulnerability. The *random* padding used has insufficient entropy and can be combined with the html sizes of the pages and the available Oracle queries to determine the user's article request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We can empirically determine that the value of com.bbn.crypto.SecureRandom.nextInt() may only return 8 possible values, depending only on the initialization of the SecureRandom instance. This instance persists throughout the attack and does not depend on the session or the user. Querying the server allows determining these numbers with a relatively small number of queries, on average less than 30 (coupon collector problem). This size gives only 8 possibilities for the size of the pad prepended to the response by ContentController.getMasterPage(). Then measuring the size of the response allows the attacker to classify the possible responses according the article size + the pad. This size is rounded to the nearest 0 mod 16 block because of the encryption. This still allows for relatively small bins (about 20 elements on average) of possible articles, which, even with 8 possible keys, allows determining the correct article in less than 300 operations with 95% probability."

Live-Collaborative Evaluation:

Draper identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.6 Poker

A.5.2.2.6.1 Question 004 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"Aprove gives <com.ainfosec.STACPoker.PokerServer: void remove(int)> LinearUpper and <com.ainfosec.STACPoker.PokerServer: void handle(int,java.lang.String)> Maybe Potentially vulnerable methods which are not problematic: PokerServer.sendMessageToClients() serializes and sends the responseMap which contains the player information, including the hole cards. However, the padding added to the map makes the message size uncorrelated with the input data.

Modulo the cryptographic security of ThreadLocalRandom.nextInt(), the method does not reveal information by space side-channel.

Potentially vulnerable methods which have (sufficiently!) sanitized input:

Obviously, PokerServer.updateClientViews() reveals information depending on the values of revealOpponentsHoleCards. However, these variables are set to the appropriate values, and the

opponentsHoleCards is set to null if revealOpponentsHoleCards == false. This prevents information relating to the opponent's hole cards from making it into the data sent to each respective user."

Take-Home Evaluation:

The challenge does not contain an intended SC Space vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.6.2 Question 031 (SC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The vulnerability involves observing the time to render the adversary's cards during the first blinds round. The time taken to render the board as sent to the target player depends on the cards being rendered. During the first blind round, only the first hole card being dealt to the player is rendered, and the bitmap is sent to the player.

The timing of this process can be measured and is precise enough to significantly diminish the search space for the two-hole cards. Vulnerable method(s): Renderer.drawTable() (Partial) Call stack to vulnerable method:

PokerServer.run() -> PokerServer.startNextHand() ->

PokerServer.runNextBettingRound() -> PokerServer.runBlinds0Round() ->

PokerServer.updateClientViews() -> PokerServer.sendTableToClient()

Reason why the vulnerable path is possible (if no demo script):

This is straightforward: the path gets executed during normal interaction.

Reason why the vulnerability is under budget (if no demo script):

By running experiments with the timing of the rendering times of various configurations, we can narrow the possibilities for card that was rendered to less than a dozen.

Doing this for both cards allows us to guess the hole with less than a few hundred tries on average."

Take-Home Evaluation:

The challenge contains an intended vulnerability in the time it takes to render a card. Following a skin change previously rendered cards are taken from cache while cards that have not been rendered with the new skin are generated and stored in the cache. The attacker can use their active operations to segment the deck evenly into previously rendered, and not previously rendered following a skin change. The attacker is then able to start a new game with the benign user and use the side channel combined with the available Oracle operations to determine the secret. Draper's response shows that they identified the timing difference, but they add that they are able to narrow the possibilities for the rendered card to <12 possibilities. We don't believe this to be possible on the reference platform/reference network environment. Draper appears to have identified the intended vulnerability, but their exploit contains some inaccuracies.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.7 Searchable Blog

A.5.2.2.7.1 Question 027 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Tool output files:

Potentially vulnerable methods which are not problematic:

`RealVector estimateStationaryVector(RealMatrix M, RealVector v, double precision)` does not terminate if, e.g., `M=[[0,0],[0,-1]]`, `v=[0,1]`, and `precision = 1.0E-7D`. However, the only invocations of this method (in `com.bbn.RankingService`) ensure that `M` does not contain negative values which, as far as I can see, ensures termination."

Take-Home Evaluation:

The challenge contains an intended vulnerability. The ranking algorithm performs power iteration; by submitting a single blog an attacker can setup the adjacency matrix for the blog site to cause the power iteration formula to perform additional iterations. This can be used to delay the runtime of a benign user's request. Draper explicitly identified the method that performs the power iteration but determined that it was not vulnerable.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"The following example.html causes a response time in excess of the operational budget when uploaded to the server: […] This page creates a cycle in the graph of links between the pages. The algorithm that creates the ranking is based on the PageRank algorithm, where the rank is computed by MatrixRoutines.estimateStationaryVector(). To avoid non-termination in the presence of cycles, a damping factor is added to the adjacency matrix, in this case defined to be 1 - RankingService.FACTOR = 0.001. As explained in "The Second Eigenvalue of the Google Matrix" by Haveliwala & Kamvar, the rate of convergence of the estimation is proportional to the inverse of the damping factor, which in this case is slow enough to allow exceeding the operational budget."

Live-Collaborative Evaluation:

Draper identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.8 SimpleVote 1

A.5.2.2.8.1 Question 019 (AC Time, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"This answer to this question is no AC time vulnerability because all user inputs (i.e. ballot answers) are sufficiently sanitized: The number of characters in user input are limited. Also, Javascript/HTML tag characters (<>) in user input are escaped, so no code injection is possible."

Take-Home Evaluation:

The challenge program contains an intended AC Time vulnerability. The attacker can cause the bounds of the registration key to be expanded and submit a key that takes a long time to process, delaying a benign user's request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.2.2.8.2 Question 025 (SC Time/Space, Unintended Not Vuln., Take-home: Yes, Live: No)

Take-Home Response:

"Method com.cyberpointllc.stac.basicvote.handler.EditBallotGuide.handlePost has an SC time vulnerability. When submitting a ballot, if one of the ballot answers has changed from the previous submission, then the server will run longer (by invoking method handlePostCoordinator) than if all of the answers where the same as the previous submission. Furthermore, the number of choices or checkboxes for a question is limited to 20. The attacker can learn by submitting more than 20 selections for an issue using script update_ballot.py. This was performed by updating variable answers in update_ballot.py. Therefore, the attacker can attempt to the ballot issue response 20 times and observe when the servers responds faster than usual. This will enable the attacker to determine when submitted responses match what is already saved.

Vulnerable method(s):

com.cyberpointllc.stac.basicvote.handler.EditBallotGuide.handlePost
com.cyberpointllc.stac.basicvote.Reply.<init>

Reason why the vulnerable path is possible (if no demo script):

Script update_ballot.py allows updating the ballot of a user.

Reason why the vulnerability is under budget (if no demo script):

At most 20 ballot update requests are needed."

Take-Home Evaluation:

The challenge program contained an intended vulnerability whereby a registration key is transformed to an integer in a deterministic manner, and the runtime for processing a registration key is linearly correlated to the value of the key. The side channel strength is dependent on the secret range, and the application code must unambiguously provide this. The intent was to have a 1-to-1 mapping between a registration key and a translated integer value of the key. However, in the application code 3 specific characters in the registration key are replaced with 3 specific numbers during the transformation from registration key input to an integer. The challenge program therefore allows for 2^6 possible registration keys to map to the same integer. There are not enough oracle queries to resolve this. This unintended not-vulnerable case was identified by Vanderbilt.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“We do not believe it is possible to find a vulnerability within the budget that allows an attacker to log in as another user. Our previous answer did not address how the attacker can log in or get the registration key of another user. There is no SC vulnerability within the budget that reveals another user's registration key because the table in class com.cyberpointllc.stac.algorithm.FunkyFunction mapping indices to registration keys can have too many collisions to invert the table within the budget. Specifically, method com.cyberpointllc.stac.basicvote.PermissionAuthenticator.confirmEssence uses FunkyFunction.funkyFunction to authenticate the registration key. Method FunkyFunction.funkyFunction uses the table field in class FunkyFunction. Field table is not invertible within the budget.”

Live-Collaborative Evaluation:

Draper responded with the same issue Vanderbilt raised during the take-home engagement. The key collisions result in the intended vulnerability exceeding the budget.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.9 STAC SQL

A.5.2.2.9.1 Question 013 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“We are looking for an input of size <=12kB which results in an output file of size >=2MB. The application writes to two files: A log file and a database file. First of all, I can't see a way to trigger the execution of several SQL commands on the server by sending a single command, i.e., every command and all data which is logged and processed on the server has to be sent to the server explicitly (and hence is limited to 12kB). The log file only contains the issued SQL commands. (In especially, as far as I can see there's no logging of Exceptions, which was the reason for at least one vulnerability in E4). It does not contain the response sent by the server, i.e., sending multiple SELECTs does not blow up the log file. Hence, the size of the log file cannot exceed 2MB with the given input budget. In the database file, all data is stored in Huffman encoding (i.e., quite space efficient). Thus, I don't see how the size of the database file could exceed 2MB with the given input budget. (The most likely spot for a missed vulnerability is the implementation of the Huffman encoding, which is rather complex and might be buggy.) After some additional analysis, the Huffman encoding looks safe. Standard algorithms are used to build the encoding and everything seems to be implemented quite straight forward. As the tree representation of the encoding has to be saved in addition to the encoded data, there exists a small blow up in size for small data. But this linear blowup is by orders of magnitude to small to make the outpult file size exceed 2MB.”

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The Huffman compression scheme can be used to exceed the resource usage limit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“For the datatype BLOB, the Huffman Trees are updated instead of rebuilt when an SQL UPDATE is executed and some of the bytes do not have a representation in the current Huffman Tree. Thus, the frequency of bytes in previous data still influences the updated Huffman Tree. This allows to create a Huffman Tree where the length of the encoding of a byte does not correspond to its frequency in the current data. This enables the following attack: First, we create a table with a single column of type BLOB. Then, we add many rows with value [0]. Afterwards, we update these rows to the values [1],...,[255]. In this way, we obtain Huffman Trees where the encoding of 0 is unreasonably long. Thus, a final update of all rows to [0,...0] (i.e., a long sequence of 0-bytes) causes the database file to grow beyond the resource limit.”

Live-Collaborative Evaluation:

Draper identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.10 Stegosaurus

A.5.2.2.10.1 Question 003 (SC Time, Unintended Not Vulnerable, Take-home: Yes, Live: No)

Take-Home Response:

“High level description of vulnerability: An attacker can observe the progress of the encoding of the secret message. Encoding a bit with value 0 is considerably faster than encoding a bit with value 1. By polling the server repeatedly, the attacker can thus guess the number of ones in short sub-parts of the message. Once this is done for all sub-parts, the attacker can reconstruct the secret message using only few queries to the oracle.”

Take-Home Evaluation:

Draper identified the intended vulnerability; however, this was determined to be insufficiently strong.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“The suspected vulnerability does not leak enough information to allow an attacker to guess the secret. While the application leaks internal information about the hiding process in Steger.hide() and there is a significant difference in the time needed to encode a zero or a one, in practice, an attacker is not able to query the server fast enough to gather enough information. The current progress of the Stegger.hide() method is exposed to the attacker via /progress.data. However, the data on this page does not directly originate from the Stegger class. Instead /progress.data displays the status information via Heartbeat.getStatus() from the Heartbeat thread associated with the job. That thread only updates the current status at most every 50ms. […]”

Live-Collaborative Evaluation:

Draper correctly determined that the intended side channel is not sufficiently strong.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.2.2.11 StuffTracker

A.5.2.2.11.1 Question 030 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“The program does not properly check if the uploaded xml files are structured as described in the `/example/SDL and Request Details.txt` file. This allows us to craft a xml file which causes the parser to run in quadratic time (unfortunately not in the question) and write large files to the file system. When we later search for an item contained in these large files, the program needs large amounts of memory to construct an answer.

Vulnerable method(s):

Reason why the vulnerable path is possible (if no demo script):

The parser which parses the input xml posted to `/stuff` does not check if the xml file is structured according to the program description. More precisely the `endElement` method in the `ItemListHandler` class writes some content to a file every time it encounters a closing `</list>` tag. It then does not clear the previously parsed data from the `this.accum` field.

On every encounter of a `</list>` tag the `createSDLDoc_and_LocalDictionary()` method constructs a string proportional to the length of `this.accum` and writes it to a file. Since the length of `this.accum` also increases by one every time, this causes the file to grow quadratically.”

Take-Home Evaluation:

The challenge program contains an intended vulnerability in the SDL formatting scheme. Draper appears to have reported an unintended vulnerability; however, after running their exploit on the reference platform/network environment, the resource usage limit was not exceeded. We don’t believe the challenge program is vulnerable to this reported unintended vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.2.2.12 Tawa FS

A.5.2.2.12.1 Question 012 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Vulnerability 1 (out of scope):

The application reads the current working directory from a cookie with name "pwd". Changing the value of this cookies to "//////////..." (~10000x) using the development tools of the browser causes the server to hang (and print a bunch of exceptions to stdout). The motivation to try this was that there are several commands like `pwd.split("/")` in the source code.

Unfortunately, the same attack still works if one changes the name of the cookie. It also works if one sets the value to some other long enough string. Thus, the problem is in NanoHTTPD, not in the server code. Hence, this vulnerability is out of scope.

Vulnerability 2:

To parse the body of a request, NanoHTTPD looks up the size of the body in the field `content-length` of the header of the request. If this value is set to `n`, then NanoHTTPD tries to parse `n` bytes from the body. If there aren't that many bytes available, then parsing the body throws a `java.net.SocketTimeoutException` after ~5s. Thus, if one sets the field `content-length` in the header to a value which exceeds the size of the body, then the server blocks for ~5s.

The method `com.bbn.RequestHandler.generateContent` does not take any precautions before invoking `getBody` when receiving a `POST` request. In especially, it does not prevent an attacker from sending arbitrarily many `POST` requests in a row, which enables an DoS attack by sending multiple `POST` request where `content-length` exceeds the size of the body.

Vulnerable method(s): `com.bbn.RequestHandler.generateContent`

Additional notes:

The size of a single attacker request generated by the provided demo script is 130 bytes. Thus, the input budget (40 kB) allows for ~300 attacker request. To exceed the resource limit (50s), less than 50 attacker requests are required.

To reproduce the attack, start the server and run the provided script with a number `n` as argument. The number `n` specifies how many attacker requests will be sent to the server.

Note that this is not a bug in `NanoHTTPD`, but in the application code. The attack exploits [a] that reading from a socket blocks for a long time (5s) and [b] there are only very few (2) threads running on the server. Both parameters are set from `com.bbn.Tawafs` explicitly (the timeout is passed to the `NanoHTTPD.start` and the number of threads is fixed by explicitly creating a thread pool of size 2).”

Take-Home Evaluation:

The challenge program contains an intended AC Time vulnerability. An attacker can cause a self-recursive structure in the defragmentation process. Draper appears to have identified two unintended vulnerabilities. Both vulnerabilities appear to be out of scope. In the case of the second vulnerability, out of scope vulnerabilities are not determined by the failure of the application code to sufficiently sanitize inputs, but by the control branch for the vulnerability being present in the *in-scope* application code. Note, teams are not penalized for identifying out-of-scope vulnerabilities; credit is still received, but other teams are not penalized for not discovering the vulnerability in the case of a null challenge question.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3 GrammaTech

A.5.3.1 GrammaTech Overview

GrammaTech answered 20 questions in the take-home engagement with a 60% accuracy. The team's accuracy did not decrease from the raw response accuracy to the reviewed response accuracy. This is an indication of the quality of their justifications in cases where they correctly identified or ruled out a vulnerability. The team had the most difficulty with AC Time questions with a 33% accuracy during the take-home. The effect of GrammaTech's more conservative approach to answering challenge questions is somewhat demonstrated by the accuracy for vulnerable vs. non-vulnerable responses and "Yes" vs. "No" responses. The team was 100% correct in their responses for non-vulnerable challenges and their "Yes" responses. This indicates that the team ensures a high degree of confidence before answering "Yes". During the live collaborative engagement, the team's accuracy increased 27 percentage-points to 87%. In instances where the team changed their answer their accuracy was 100%.

In the take-home engagement, GrammaTech were close to the intended vulnerability on 2 questions IBASys Question 029 and Searchable Blog Question 027. The team correctly answered two these questions correct following collaboration. Poker Question 004 presented an interesting collaboration opportunity as GrammaTech and Colorado identified a weak side channel and ruled correctly that it was not sufficiently strong. Vanderbilt and Iowa had identified similar potential side channels but ruled them sufficiently strong. The decision point for the presence or absence of a vulnerability was due to whether the random seed used to pad packets could be determined. As the result of collaborations, all teams correctly concluded that there was no way to determine the random seed for the benign user and predict the padding sequences, as separate *Random* objects are used in the benign user and attack threads and the challenge question limits the attacker to a single game with the benign user. This was a highlight of what was expected from the collaborative engagement.

A.5.3.1.1 Take-Home Engagement Scores

Table A-89: Engagement 5 GrammaTech Take Home Engagement Question Acccuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	6	4	67	4	67
SC Time	5	3	60	3	60
SC Space/ Time	0	0	0	0	0
AC in Space	6	4	67	4	67
AC in Time	3	1	33	1	33
Total	**20**	**12**	**60**	**12**	**60**

Table A-90: Engagement 5 GrammaTech Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	16	8	**50**	8	**50**
Not Vulnerable	4	4	**100**	4	**100**
Yes Answer	8	8	**100**	8	**100**
No Answer	12	4	**33**	4	**33**

A.5.3.1.2 Live Engagement Scores

Table A-91: Engagement 5 GrammaTech Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	6	4	67
SC Time	5	4	60
SC Space/ Time	0	0	0
AC in Space	8	8	100
AC in Time	4	4	100
Total	**23**	**20**	**87**

Table A-92: Engagement 5 GrammaTech Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	19	16	**84**
Not Vulnerable	4	4	**100**
Yes Answer	16	16	**100**
No Answer	7	4	**57**

A.5.3.2 GrammaTech Responses

A.5.3.2.1 Accounting Wizard

A.5.3.2.1.1 Question 011 (SC Time, Unintended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We believe (with low confidence) that no side-channel vulnerability in time is present.
Statements on if and how the observables relate to the secret

The secret for this question is the expenditure limit for a project. The running time of some requests can be affected by the costs of the budget lines in a project's budget and the expenditure limit for that project. In particular, if the total cost of all budget lines exceeds the expenditure limit, then the com.bbn.accounting.budgeting.BudgetSolver class is used to perform a (sometimes slow) search for a subset of budget items that fit within the expenditure limit. If the

total cost of all budget lines is less than the expenditure limit, then the slow search is not performed.

Statements on any potential side channels within the application

An attacker can repeatedly update the number of work hours that they have expensed to a particular project, and observe the amount of time taken to respond to this query, or the amount of time taken by a project manager's expenditure report query. If the time is relatively low, then the attacker can conclude that either (1) the project's total cost is less than its expenditure limit or (2) that the total cost of all hours worked by all employees on this project is greater than the project's expenditure limit. If the time taken to respond to a query is very high, then the project's total cost must lie between those two values. The attacker can also increase the time that the server takes in the slow-response case by first expensing several items to the project. Some information is also leaked by the length of the responses to project managers' expenditure report queries.

Statements on why the application does not contain a side channel (strength and reachability).

The timing attack described above only suffices to find the so-called "adjusted expenditure limit", which is the project expenditure limit minus the total cost of all the work hours that employees have charged to the project. Note that the attacker is able to send a request to determine the number of hours that the attacker himself/herself has charged to the project. And, the attacker can also remove their own expensed items and other employees' expensed items from the project by exploiting a bug in the code that responds to the item deletion query: if the attacker requests that an item named "foo" be deleted from the project, but there is no item named "foo", then the code will delete some other item instead; in this way, an attacker can delete all items from the project, even though they cannot actually see the items that were added by other employees. Importantly, however, these techniques provide no way to determine the number of work hours that other employees have charged to the project. As a result, the above-described techniques only allow the attacker to attacker find the adjusted expenditure limit, not the total expenditure limit. An attacker might also try to take advantage of the information leaked by the length of expenditure reports. Importantly, an expenditure report contains a total field for each project, which provides information about the total cost of all budget lines for the project, including any hours charged to the project by other employees. The attacker cannot directly query for expenditure reports, because the attacker only has employee credentials. Nor can an attacker read a project manager's expenditure reports by passively observing network traffic, because expenditure reports are encrypted with HTTPS. However, because the expenditure report is communicated in JSON format, all numerical values in the report are stored as text, so, different numerical values are expressed using different numbers of digits. In particular, if an attacker could reliably discover the length of the total field of an expenditure report while repeatedly manipulating the cost of some budget lines, they might be able to find a point at which the total field became one digit long, which would happen when the total was between 0 and 9 dollars. In other words, by sending a series of queries with different (negative) item costs, the attacker could learn the current total cost of all budget lines to within 10 dollars. This would include the cost of employees' hours worked, which is the difference between the expenditure limit and the adjusted expenditure limit. Then, by combining this technique with the techniques described above for learning the adjusted expenditure limit, an attacker could find the expenditure limit for a project. In experiments, however, we were not able to reliably discover

the length of an expenditure report by observing HTTPS traffic, and without that information, this attack cannot be completed.

How our tools supported this conclusion

We have heuristics that identify regions of the code that are suspected of having high time complexity. One of these regions (ranked 4th out of 29 regions identified) consisted of the methods BudgetSolver.vectorDec, BudgetSolver.getAffordableLines, and BudgetSolver.getAllowedExpenditures, which have running time that is related to the expenditure limit, as described above. We used our call-graph visualization to assist in manual inspection of the code."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Employee.updateHours is a method that is called when an employee tries to log hours for him or herself. If an employee does not have an associated hours object then FileStore.newHours will be called and a new hours object will be created and the associated project file will be updated. Within FileStore.newHours, FileStore.hoursFit is called which simply compares the new hours amount with the project expenditure. If the amount of new hours is more than the project expenditure then FileStore.hoursFit will return false, and FileStore.newHours will throw an over budget exception and no other logic from FileStore.newHours will execute. However, if the hours do fit and FileStore.hoursFit returns true, then the new hours object will be written, and the associated project will be written. If an attacker is able to distinguish the amount of time it takes to write an hours and project object, he or she will know whether or not their hours are below the project expenditure limit or not. This distinguishing capability would allow an attacker to perform a binary search attack by updating their hours […]"

Live-Collaborative Evaluation:

There was a mistake in packaging this challenge program: the application came provided with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects' contents or states.

The unintended side-channel vulnerability identified by GrammaTech and several STAC teams was confirmed.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Statements on any potentially vulnerable code structure in the application
Because this question asks about files stored on disk, we investigated several parts of the code that write to disk. This included the four types of com.bbn.accounting.objects.BaseObject written to the com.bbn.accounting.objects.FileStore, namely Budgets, Projects, Hours, and Items, as well

as the parts of the code that write to the log file.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures
The file associated with a Budget on disk contains the UUID of the Project objects that are children of that Budget object; likewise, the file associated with a Project contains the UUIDs of the Items and Hours objects that are children of that Project object. The file associated with a Project object will grow in size when an attacker expenses items to a project, because more UUIDs must be stored. However, we determined that the project file size only grows linearly with the number of items added. We did not discover any way to exploit the code that writes to the log file to cause a large amount of data to be written. We investigated the use of language localization by the logging system but did not find exploitable behavior.

Statements on why the application does not contain a complexity vulnerability (strength and reachability)
The attacker is able to increase the size of some FileStore files (mainly the serialized form of Project objects) by adding items to a project. However, we determined that the input budget was not sufficient to cause a large increase in the size of such a file.

How our tools supported this conclusion
We have heuristics that identify regions of the code that are suspected of producing a large amount of output. The highest-ranked such region consisted of the methods ProjectManagerManager.setProjectExpenditure, FileStore.writeFileStore, FileStore.newHours, FileStore.newItem, EmployeeManager.updateHours, EmployeeManager.orderItems, BaseObjectMarshaller.toBytes, and BaseObject.write. These methods produce logging output and output to the file store. We used our call-graph visualization to assist in manual inspection of the code."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attempt to switch the local to and unsupported on triggers verbose logging mode that can be used to submit inputs that cause the server to exceed the resource usage limit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"When LocaleNotSupportedException is initialized it sets the application's log level to trace, which is much more verbose than the default level. Specifically, when the log level is set to trace, any calls to BaseObject.uuid prints the base object's uuid to the log file. When BaseObjectMarshaller.toBytes is called with an object that has children, BaseObject.uuid is called on all those children. Furthermore, adding a budget item to a project causes BaseObjectMarshaller.toBytes to be called on that project. Thus, adding a new budget item will cause the application to write the uuid's of all previous budget items associated with that project. Therefore, adding a new budget item when the log level is trace causes quadratic blow-up. […]"

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.1.3 Question 023 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The code that is responsible for this vulnerability is mostly contained in the com.bbn.accounting.budgeting.BudgetSolver class, and particularly in the getAffordableLines method of that class […]

Why the code contains a vulnerability

When the total cost of all budget lines for some project exceeds that project's expenditure limit, the BudgetSolver class is used to perform a slow (exponential time) search for a subset of budget items that fit within the expenditure limit. In particular, the while loop in BudgetSolver.getAffordableLines can run for exponential time. Given a budget having N non-required budget lines, the while loop considers up to 2^N different subsets of the set of budget lines. This code is called in response to the benign request described in the problem (i.e., an expenditure report requested by a manager).

Why the vulnerability is exploitable

The attacker has employee credentials. Therefore, the attacker can send a series of requests that add many expensive items to the project budget. Ideally, the attacker would choose a price for these items such that any single item fits within the project's budget limit, but no more than one item will fit. In this situation, the code for BudgetSolver.getAffordableLines will search through exponentially-many subsets of the added items before finding a subset of items that fits the budget. The attacker faces one difficulty when performing this attack, namely, how to choose the correct price for the items to be added to the budget. The BudgetSolver.getAllowedExpenditures method defines the "adjusted expenditure limit" to be the project expenditure limit minus the sum of all employee wages for that project. Given a project with an adjusted expenditure limit of L, the attacker needs to choose a price for their items that is between L/2 and L, so that each item, individually, will fit the budget, but no two items will fit. Items must have prices no greater than L because the code explicitly ignores items with prices greater than L, therefore such items cannot contribute to the exponential running time. The attacker cannot directly set the adjusted expenditure limit, nor can they directly observe it. However, they can have an effect on the adjusted expenditure limit by sending an /updateHours/ request, which allows them to report the number of hours that they worked on this project; moreover, the attacker can report a negative number of hours worked. Thus, the attacker can choose a very large number X of dollars (for example, 10 million) and charge enough negative hours so that the adjusted expenditure limit increases by X. If the attacker then charges many items that each cost X/2 dollars, the code will be forced to perform an exponential-time search. (Note, however, that this attack makes two assumptions: that the project is not already over its budget by X/2 dollars, and that the project initially has an adjusted expenditure limit of less than X/2. These are relatively weak assumptions because the attacker is free to choose a very large X. The code sets an upper limit of 3 million dollars on the un-adjusted expenditure limit of a project, so an X value of 10 million seems like a reasonable choice, although there are circumstances-- such as a project that is massively over-budget, or a project in which another employee has charged many negative hours-- in which that value of X would not work.) We implemented this attack and found that 29 items were sufficient to exceed the 500 second running time, and they required 24 kilobytes of requests (as measured using tcpdump).

How our tools supported discovery of the vulnerability

We have heuristics that identify regions of the code that are suspected of having high time complexity. One of these regions (ranked 4th out of 29 regions identified) consisted of the methods BudgetSolver.vectorDec, BudgetSolver.getAffordableLines, and BudgetSolver.getAllowedExpenditures, which contain the key parts of this vulnerability. We used our call-graph visualization to assist in manual inspection of the code. We manually constructed an exploit script."

Take-Home Evaluation:

GrammaTech identified the intended vulnerability

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"How the observables relate to the secret
A possible observable for a user with employee credentials is the sizes of files associated with the application on disk. If the employee asks for the sizes of files, the application will report back the number of files of a certain size for all files on disk. As far as the basic application, there are 4 different objects that are written to disk: Budget, Project, Hours, and Item. All four of these objects extend com.bbn.accounting.objects.BaseObject. Each BaseObject has set of children which are other BaseObjects. Thus BaseObjects represent a tree data structure. A Budget's children are projects. A Project's children are made up of Hours and Items. Hours and Items have no children. This question asks if an employee can determine the number of Hours and Items that are children of a Project he or she works on. Each of the four objects are written to the server's disk, each in separate files. The name of these files are generated uuid's with constant length. For every object with children, the uuids of the children are written to the file for that parent object. The important object for this question is the Project object. The Project object has two fields that have variable size: the name of the project and the number of children associated with that project. This means a Project's file size is completely determined by the Project name, which is known to the attacker, and the number of children the Project has, which is what the attacker wants to know. Furthermore since uuids are of constant length the size of the Project file is a constant factor of the number of budget items. Thus, if an attacker can determine the size of the Project file of interest, he or she can determine the secret in question.

Why the observables reveal the value of the secret

Based on the above reasoning the attack focuses on determining the size of the Project file of interest. Any employee can make a request to see the sizes of all the files on the server. Thus, the attacker simply logs on and requests to see the sizes of all the files. The server responds by giving count and size pairs to the attacker. The attacker then knows the sizes of all the files, but it

doesn't know which size is associated with the project of interest. He or she can isolate the size of the project file by modifying it. If he or she adds an item to the project of interest and requests to see the files again, the result will be different in two ways. First, the new item will be in the size counts. Second, the parent project file size will change. This will allow the attacker to identify the size of the project file. Once the size of the project file is known, and since the attacker knows the name of the file, he or she can determine the number of budget items and succeed with the attack. The number of operations required is 5 operations in total: 1 for logging in, 1 for the first file size snapshot, 1 for adding a new item, 1 for the second file size snapshot, 1 oracle query.

Why this side channel is triggerable by an attacker

The attacker for this question is a user with Employee credentials. The documentation specifies that an Employee can add and remove items for a project they are working on. An Employee can also ask for file sizes. These are the only operations needed for the attack as described above.

How our tools supported discovery of the vulnerability

We have heuristics that identify regions of the code that are suspected of producing a large amount of output. One of these regions (ranked 1st out of 3 such regions identified) consisted of the methods ProjectManagerManager.setProjectExpenditure, FileStore.writeFileStore, FileStore.newHours, FileStore.newItem, EmployeeManager.updateHours, EmployeeManager.orderItems, BaseObjectMarshaller.toBytes, and BaseObject.write. These methods produce logging output and output to the file store. By manual inspection, we determined that the vulnerability depended on the writes to the file store. We used our call-graph visualization to assist in manual inspection of the code. We manually constructed an exploit."

Take-Home Evaluation:

GrammaTech identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.2 BattleBoats 2

A.5.3.2.2.1 Question 010 (SC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"How the observables relate to the secret

The observable we use is the time taken by the victim machine to reply to an attacker message of type "SHOT_MADE". When receiving a "SHOT_MADE" message, the victim's machine computes and sends a "HIT_RESULT" message to the attacker machine. In order to compute the hit results, the victim's machine uses the position of the cannon and the shot parameters to compute the set of ocean squares targeted by the shot. The most interesting method is

pullOceanSquares in the class com.cyberpointllc.stac.combatboat.OceanPanel. If the shot happens to be out of the bounds of the ocean board, the methods this.takeHits and this.updateHitsAsSunk are executed twice, otherwise they are executed once. These two methods have significant time cost. Other than this method, the time cost to send the "HIT_RESULT" message is about constant.

Why the observables reveal the value of the secret (even in the worst case) with a side channel that is sufficiently strong, and why this side channel is triggerable by an attacker

We believe that the attacker is able to learn the position of its cannon using the time information explained above. When playing against the victim, he or she can use a binary search strategy to learn the horizontal and vertical positions of the attacker's cannon: first, the attacker can send a horizontal shot that travels half the length of the board depending if the HIT_RESULT message is received in a shorter or a longer time, the attacker learns that the shot was or was not out of bounds on the ocean board, and thus knows if the cannon is horizontally in the first or second half of the board For the next shots, the attacker can continue the binary search strategy and can learn both the vertical and horizontal position of the cannon in less that 10 shots (because the board length is at most 20).

How our tools supported discovery of the vulnerability

Several of our filtering heuristics identified method com.cyberpointllc.stac.combatboat.HitLocator.computeHitTime() as suspicious. The vulnerability was discovered by manually browsing the source code starting from this and related methods."

Take-Home Evaluation:

GrammaTech identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.2.2 Question 021 (AC Space, Intended Vulnerable, Live: Yes)

Live-Collaborative Response:

"The vulnerability is in the numerical computations associated with calculating the hit time of a cannonball shot. The program computes this by finding roots of the relevant quadratic equation using Newton's method. The following is the main loop that runs Newton's method […]

Once the number of iterations exceeds HitLocator.ITERATIONS_TO_ALLOW in computeHitTime, the TrajectoryData objects are stored in the hashmap trajectory for every subsequent iteration to allow termination in case of a cycle. […]

During the take-home engagement, our tools identified computeHitTime as a method with potential high time complexity, and we noted that it operates on BigDecimals which made us

suspicious about its heap space utilization in the worst case. However, we were unable to craft an attack during the take-home engagement and so we did not answer this question.

During the collaborative engagement, we worked with the Utah team and refined our understanding of the above code as well as potential edge cases with the use of very small BigDecimal values. We used profiling to further refine our understanding of the execution patterns and memory allocations in the code. This allowed us to craft an attack that exceeded the budget specified in the question."

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.3 BraidIt 1

A.5.3.2.3.1 Question 007 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"How the observables relate to the secret

We understand the secret to be a number between 1 and 5, representing the number of the braid which was modified. This is stored as field plaitCount of com.cyberpointllc.stac.plaitthis.phase.PlaitSelectedPhase. The value of this field is set when the user runs the select_braid command in com.cyberpointllc.stac.plaitthis.command.SelectPlaitCommand.execute(). The observable is the size of network messages. The observable relates to the secret because the value of plaitCount is used as a parameter to specify the width of the string representation of the modified braid. The relevant code snippet is as follows, found in com.cyberpointllc.stac.plaitthis.phase.PlaitSelectedPhase: […] The string returned from this method is used in constructing the message which is sent across the network to the player (in this case, the attacker), in com.cyberpointllc.stac.plaitthis.command.TransmitModifiedPlaitCommand.TransmitModifiedPlaitCommandCoordinator.invokeExecutor: […]

Why the observables reveal the value of the secret (even in the worst case) with a side channel that is sufficiently strong

Once the attacker has the message constructed by the above invokeExecutor() method, it is straightforward to extract the portion corresponding to the modified braid and divide the length of this string by 50. This yields the index of the modified braid, which is the secret.

Why this side channel is triggerable by an attacker

Since the information about the secret is directly encoded in the (length of the) response to the attacker player, the attacker can simply process the response that he or she receives. The allowed budget of 12 operations is well within the normal number of interactions between the attacker and victim player that occur within the course of a normal game.

How our tools supported discovery of the vulnerability

Our tools identified com.cyberpointllc.stac.communications.CommunicationsConnection.write and com.cyberpointllc.stac.console.Display.executeCommand as methods that generate output

which is sent across the network. The vulnerability was discovered through manual browsing of the decompiled code starting from the above methods."

Take-Home Evaluation:

GrammaTech identified an unintended vulnerability related to the intended one.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.3.2 Question 017 (AC Space, Intended Vulnerable, Live: Yes)

Live-Collaborative Response:

"Plait.simplifyOnce has a vulnerability in that it will not simplify some braids that can in fact be simplified. Specifically, Plait.simplifyOnce calls Plait.obtainEnsuingPortion, which modifies a substring of the braid which is flanked by inverses. If Plait.obtainEnsuingPortion returns null then Plait.simplifyOnce performs no modification to the braid. Since there are some braids that Plait.simplifyOnce does not simplify, it is possible to get Plait.simplifyCompletely to go into an infinite loop. […]

In short, this method simplifies braids. It does this by using Plait.freeSimplify and Plait.simplifyOnce. Plait.freeSimplify will modify a braid by looking for adjacent inverses, whereas Plait.simplifyOnce will look for non-adjacent inverses and modify the braid to allow for free simplification. However, if there exists an unsimplified braid which Plait.simplifyOnce does not modify then Plait.simplifyCompletely will execute an infinite number of times and will grow the log file at each step. […]"

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.4 BraidIt 2

A.5.3.2.4.1 Question 001 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"How the observables relate to the secret

We understand the secret to be a number between 1 and 5, representing the number of the braid which was modified. This is stored as field weaveCount of com.cyberpointllc.stac.braidit.phase.WeaveElectedState. The value of this field is set when the user runs the select_braid command in com.cyberpointllc.stac.braidit.command.SelectWeaveCommand.execute(). The observable is the size of network messages. The observable relates to the secret because the value of weaveCount is used as a parameter to specify the width of the string representation of the modified braid. The

relevant code snippet is as follows, found in com.cyberpointllc.stac.braidit.phase.WeaveElectedState: […] This method generates a String whose size is known from the value of this.weaveCount * weaveStr.length(). When the victim sends its modified braids using the command send_modified, it builds a message and sends it to the attacker. The construction of the message is in the method com.cyberpointllc.stac.braidit.command.executeService. The message contains the following information:

- the String generated by the above grabWeaveString() method.
- the String representation of the braids 1,2,3,4 and 5

Consequently, the observable (the packet size) allows the attacker to know the total length of these String values.

Why the observables reveal the value of the secret (even in the worst case) with a side channel that is sufficiently strong

The attacker is a player in the game and plays against the victim player. Consequently, it receives the packet and can display the 5 initial braids and the new braid using the print command. From there, the attacker knows the length of the String representations of all the braids (including the modified braid). Given the size of the packet it has received, the attacker can then deduce the value of this.weaveCount, because it already knows the value of weaveStr.length() in grabWeaveString().

Why this side channel is triggerable by an attacker. Note, if the side channel is believed to be passively persistent, a statement to this effect is sufficient

Since the information about the secret is directly encoded in the (length of the) response to the attacker player, the attacker can simply process the response that he or she receives. The allowed budget of 12 operations is well within the normal number of interactions between the attacker and victim player that occur within the course of a normal game.

How our tools supported discovery of the vulnerability

Our tools identified com.cyberpointllc.stac.comms.CommunicationsConnection.write as a method that generates output which is sent across the network. The vulnerability was discovered through manual browsing of the decompiled code starting from the above methods."

Take-Home Evaluation:

GrammaTech identified an unintended vulnerability closely associated to the intended one.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.4.2 Question 022 (AC Time, Intended Vulnerable, Live: Yes)

Live-Collaborative Response:

“This method is similar to the obtainEnsuingPortion method of braidit_1, but the difference in the condition $endIndex - startIndex < 104$, while it was 25 for braidit_1. This means that the same attack would work with a modified braid with some good properties. Because some simplification operations actually increase the length of the string representation of the braid, we believe there is a possibility to craft a braid with $length \leq 52$ (which is the maximal authorized size), that would actually expand during the simplification process to a string with $length > 104$, and with an a as the first letter and an A as the last letter. From there, the same behavior as in braidit_1 Q017 would happen, and the simplification would enter an infinite loop.

One of the possible expansions of braids during simplification follows this pattern: $a[w]b[w']A -> a[w]'Bab[w']'$ with w and w' words without a or b. Notice that we duplicate a inside the string. If we nest such pattern, we can trigger this increase several times in a row.”

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.5 IBASys

A.5.3.2.5.1 Question 015 (SC Space, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“How the observables relate to the secret, and potential side channels

The relevant output-generating methods are those that produce network traffic. Our tools identified that when executed in server mode, the only code location that creates traffic is the call to java.nio.channels.SocketChannel.write(Ljava/nio/ByteBuffer;)I in com.ainfosec.ibasys.server.IBASysServer.init([Ljava/lang/String;)V. We used both the call graph and the output warnings in the CodeSonar visualization to get this information, and determined that all detected output-generating statements except that one were irrelevant. Monitoring the network traffic using Wireshark confirmed that the server sends only one packet when a normal client executes IBASys in client mode. There can be an SCL-S only if the size of the packet (as a ByteBuffer) sent by the IBASys server depends on the secret data (e.g., the passcode image of target user).

Why the application does not contain a side channel (strength and reachability)

By exploring the code in the function com.ainfosec.ibasys.server.IBASysServer.init in the CodeSonar UI, we were able to determine that the packet contains the following in case there is no exception:

- The symmetric AES key used by the IBASys server and client, which is 16 bytes (128 bits)
- The initial vector which is 16 bytes
- The returned session key (16 bytes) that the client can use later
- The client name

The first three components are encrypted using the symmetric AES key used by the IBASys server and client. Thus, observing the size of packet on the network does not give the attacker any information about the secret data. If the passcode image checker (the class

ImageMatcherWorker) throws an exception during the matching process, then the packet sent by the server also includes the first exception message. The size of the exception message is 46 bytes. Thus, the attacker can determine whether or not an exception was thrown. However, we did not find a way to exploit this information to achieve the desired attack. Consequently, we believe there is no SCL in space that would allow an attacker to impersonate another user by just knowing their username. To further validate our answer, we used Wireshark to monitor the size of the packets received by the attacker, in case the password image is valid, and in case it is not. We saw no difference in the size of the packets, which is in line with our explanations above.

How our tools supported this conclusion

IST and its CodeSonar user interface were used to identify code locations that create network traffic, and to make sure we did not miss any user code that creates traffic (so that we can provide a 'no' answer). Once the CodeSonar warnings pointed us to the init method, manual inspection of the code led us to our answer."

Take-Home Evaluation:

The challenge contains an intended vulnerability. A race condition can be used to leak individual bits in the passcode string. This can then be used to generate an image that will pass validation.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.3.2.5.2 Question 024 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"We have identified a possible side channel leak in the method

com.ainfosec.ibasys.server.ImageMatcher.ImageMatcherWorker.run(). This method contains a loop that compares two arrays:

- this.imagedata, an array whose contents are computed from the password image provided by the client
- ImageMatcher.this.pcode, which is actually the secret, and is used to verify the correctness of the this.imagedata array, i.e. the correctness of the password image. […]

this.imagedata is previously assigned the value of pcodetest, which is an array of 128 bytes. This means that the loop is executed between 0 and 32 times. If the attacker manages to provide a password image such that the loop executes 32 times without entering the !r branch of the if condition, then the victim user is impersonated. The condition r compares the parity of the two arrays at the same index, so the attacker only needs to satisfy 32 successive parity comparisons with the secret array. Moreover, the following additional features of the pcodetest array make the attacker's job easier: the input image is first scaled so that its height and width are exactly 100 pixels. Without loss of generality, the attacker can then use images of size 100x100, and the

scaling will return the same image. The image is translated into an array of integers imageDataBuff, using the method BufferedImage.getRGB. This array of integers always has size 10000, i.e. 100x100. There is a loop that sets all indexes of pcodetest. The value of csize is always 10000/128 = 78. […] This means that for all ii between 0 and 127, pcodetest[ii] = (byte)(imageDataBuff[78*ii] % 2). Putting all the above together, the comparison operation in com.ainfosec.ibasys.server.ImageMatcher.ImageMatcherWorker.run() is insecure against an attacker measuring the time taken by the routine to return this.success true or false. The verification of a valid image takes longer than an invalid one, since the routine returns false as soon as two bytes differ. Based on this, the attacker can mount the following attack by choosing a password image of size 100*100 pixels.

For n = 0, 1, the attacker proposes 2 images whose 78*0-th pixels are n, respectively and all other pixels are 0. The attacker determines the image with the longer running time among the two attack images. The image with the longer running time provides the correct pixel at position 78*0. Once the correct pixel at position 78*0 is known, the attacker repeats the attack for 78*ii with ii = 1,...,127 until all 78*ii pixels are recovered.

Why the application does not contain a side channel (strength and reachability)

The attacker's budget is limited to a maximum of 3500 packets. The size of a password image must be at least 340,000 bytes and each individual packet has to contain exactly 10,000 bytes of the image. Therefore, the attacker needs at least 34 or 35 packets each time he or she tries to login with a manually crafted password image. Therefore, that the attacker can try about 100 different password images of size ~340,000 bytes, and then issue an oracle query to see if it was able to impersonate the victim. With only 100 attack images available, the attacker can determine at most 50 pixels of the image, so we believe the attack is not feasible within the budget required. Other attack strategies may be possible that (probabilistically) require fewer than 2 images to determine each pixel, but we believe the budget is too small to allow any such attack to succeed with 95% probability.

How our tools supported discovery of the vulnerability

Our initial heuristics that identify code fragments with high and/or variable time complexity identified com.ainfosec.ibasys.server.ImageMatcher.ImageMatcherWorker.run() as a method of interest - it was the 3rd-highest ranked by those heuristics. Starting from this methods, we discovered the vulnerability by manual inspection of the code in the IST/CodeSonar user interface."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. GrammaTech flagged a red-herring looping structure as potentially vulnerable and correctly concluded that the inability to get independent runtimes for each loop prevents a leak.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"To further justify our 'No' answer from the take-home engagement, it is possible to experimentally evaluate whether the time differences when executing the loop 1 to 32 times is noticeable in a real environment. Experimental results from other teams showed that the loop executes too fast to be exploitable as a side channel."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.5.3 Question 029 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We used our tool IST and its CodeSonar interface to identify the output-generating methods in the program. For this question, we identified that there is only one output method that writes to a file assuming that the server was NOT started with the option -Dimgscalr.debug=true. The interesting method call is the call to writeObject in com.ainfosec.ibasys.org.imgscalr.Scalr.log(ILjava/lang/String;[Ljava/lang/Object;)V. If the server is not running in debug mode, the only time the log method is called is in com.ainfosec.ibasys.server.IBASysServer.init(), when catching an exception: […] It might be possible for the attacker to send packets that would trigger this exception multiple times; this might generate a file that exceeds the size budget. We observe that the Scalr.log method writes to a file only if its parameter session is not null. For this reason, the exception must necessarily occur in the call to svr.lm.parsePacketPart.

Statements on why the application does not contain a complexity vulnerability (strength and reachability)

We determined there is a way to trigger the exception so that Scalr.log is called and writes the value of session.toString() to an XML file in the /log directory. In com.ainfosec.ibasys.server.LoginManager.parsePacketPart, the code parses the data in the received packet and, in case its content ends with '00000', it executes a loop that reads the data array backwards and decrements an integer totall until a value different from zero is found: […] If the data1 array is full of zeros, then total will be equal to 0 and the access to data1[totall-1] will trigger an ArrayIndexOutOfBoundsException.

However, the attacker does not know the other usernames in the system. Therefore, he or she can only create a single file in the /log directory, because the name of the file is the hashcode of the username followed by log.xml. There is no way to create a second file using another username, because if the username is not valid, then creating a LoginSession object with that username will fail. For that reason, we only need to verify whether it is possible to create a log file larger that the budget of 150MB. The contents of this file is an XML encoding of the session.toString() object, i.e. session.databuff.toString(). session.databuff is directly the contents of the image that the attacker sent. So, the size of the log file is linear in the size of the image sent. We implemented the attack and ran it using an image of size 1.3 MB (which is greater than the input budget), and obtained a log file of size ~22MB, which is much less that the resource usage limit of 150MB. For that reason, we believe the answer to the question is 'no'.

How our tools supported this conclusion

IST and its user interface in CodeSonar quickly led us to the call to Scalr.log in com.ainfosec.ibasys.server.IBASysServer.init()."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. Persistence of the session object can be used to bypass the input guards to trigger an exception resulting in excessive logging. GrammaTech was close to the intended vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“We answered 'No' during the take-home part of the engagement. Our justification was that, even though we had found an attack, which triggers an exception in IBASysServer.init and calls the log method, we were not able to create more than a single file. The reason was that the name of the log file that is created depends on the hashcode of the session. However, we missed the fact that the hashcode of the session is actually the hashcode of the username, and the username does not matter to successfully obtain a LoginSession object: the only valid information one need to send to get a LoginSession are the AESKey and the iv. An attacker can then use its own valid AESKey and iv, but change the username to create not just one log file but multiple files. This will eventually exceed the budget of 150Mb. We were able to slightly change our previous attack by using different names and we successfully reproduced the vulnerability.”

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability through collaboration

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.6 Medpedia

A.5.3.2.6.1 Question 009 (SC Space, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“Statements on if and how the observables relate to the secret

In this question, the attacker knows the size of the packets that the victim sends to/receives from the server. We understand the secret to be the name of the wiki page accessed by the victim. When the user sends an query to request the page, he or she sends a packet whose size depends on the length of the URL of the page, which directly depends on the page name. Because the wiki contains 48164 pages, this side channel is largely insufficient to learn the secret. When the server sends the requested HTML page back to the victim, this also creates network traffic. The relevant application code is the method com.bbn.ContentController.getMasterPage(): […]

Statements on any potential side channels within the application

There could be a side channel if the content that gets written to the object response in the above method allows the attacker to guess which page was requested by the victim.

Statements on why the application does not contain a side channel (strength and reachability)

The content of the wiki page is read from the ZIM file and stored in final ByteArrayOutputStream baos. Then, baos is written to response. The server also adds an HTTP header "X-Padding", which is set to a random alphanumeric String of a random size: […] The random number generator random is an instance of the SHA1PRNG implementation which is believed secure and is initially seeded with a random number coming from /dev/urandom. Consequently, we expect this random number generator to be sufficiently secure to prevent the attacker from guessing the size of the page that was requested. In addition, because of the budget

limit (300 operations), the attacker is unable to know the size of every page in the wiki (>48k pages): even if the attacker could learn the exact size of the requested page, there is a low probability of discovering which among the wiki pages has this size within 300 operations.

How our tools supported this conclusion

Because of the use of the Spring framework, our tools were not able to produce an acceptable call graph for this program. Consequently, we were not able to apply our automated analyses, and we proceeded through manual inspection of the decompiled source code."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The random call does not have sufficient entropy. The attacker can determine the possible paddings using active operations and use the relationship of html page size to article to determine the secret.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"During the live engagement, we collaborated with teams from Vanderbilt, Northeastern, Two Six Labs and Iowa State. They made us aware of several side channels that exist within the application. In particular, there is a side channel allowing the attacker to learn the size of the article requested by the victim, and another allowing the attacker to learn the size of the name of the requested article. We believe that those side channels exist, however, in both cases, they were (to our understanding) discovered empirically through black-box investigations rather than through analysis of the code. Because of this, and because of the numerous sources of noise for both channels, we do not feel comfortable asserting that the strength of the channels is sufficient for the attack to succeed within the specified budget. Consequently, we wish to retain "no" as our answer. We do acknowledge that our earlier statement about the security of the random generator is in error, as we have seen empirical results from other teams that show the random generator converges to a state where it generates only 8 numbers."

Live-Collaborative Evaluation:

GrammaTech observed the intended vulnerability but was not sufficiently confident following collaboration to change their response.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.3.2.7 Poker

A.5.3.2.7.1 Question 004 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"The attack must be performed once all hole cards have been dealt and before any betting starts, which means that most of the rounds of poker have not taken place yet. This greatly restricts the code that can be leveraged for the attack, unless we can use some state that persists from one hand to the next. The only state that persists across poker hands is the random number generators (RNG) (one for padding messages and one for shuffling the card deck) and the deck of cards. As the deck is shuffled between hands, we cannot use it to retrieve any information. On the other hand, obtaining the state of one of the RNGs may allow us to determine all cards in the deck (for the deck shuffling) or the size of the opponent messages (for the padding).

Statements on if and how the observables relate to the secret

The opponent's cards are sent to the other player with all game status messages. Thus, the size of the messages could be a side channel that allows the attacker to obtain the length of the strings describing the opponent's cards. This is mitigated through the use of padding of 5000 to 95000 bytes, performed by the sendMessageToClients method, which is used for all communication with clients.

Statements on any potential side channels within the application

The RNG used for padding is not cryptographically secure. It is based on the class ThreadLocalRandom from the Java library: […] The seed for this random generator has a 64-bit length and is thread specific. The next seed is computed simply by adding a constant GAMMA = 0x9e3779b97f4a7c15L. The generated number is produced from the seed using two XOR operations and two multiplications: […] before being finally taken modulo the range of value we want (here, 95000 - 5000 = 90000). There is a also rejection mechanism to avoid being unfair but it has a very low probability of being triggered (lower than 0.002%) so we can just ignore it in regard to the number of random values generated. These operations are invertible: the modular division amounts to a multiplication with the modular inverse, which can be computed offline with the Chinese remainder theorem; the XOR operation is its own inverse. Thus given padding of ten consecutive messages (which can be triggered by "listOfSkins" calls for instance), we can guess the remaining bits at the exit of the mix32 method, right before the right shift and cast to int. Inverting the rest of mix32, we can get back to a candidate seed value. Generating the next values from this candidate and comparing them to the actual next values observed, we get the actual seed. Since 2^16 < 90000 < 2^17, we initially have 16 bits of information and we want to guess 64 - 16 = 48 bits. Accounting for duplicate information that may arise in consecutive RNG calls, the number of random values required for that should be about 10 (coupon-collector problem). The naive computation required to test these 48 bits lasts several days; however, there may be a more sophisticated way the get the seed.

Statements on why the application does not contain a side channel

Once we get the length of the string describing each opponent card, we can use oracle queries to test all possible pairs of cards. The worst case is length 7, which corresponds to 17 possibilities: 2 to 9 of hearts or spades and 10 of clubs. In the worst case, trying all 17 possibilities for both cards requires 289 trials which is greater than the allowed budget (175). Furthermore, the time taken to (naively) obtain the seed spans several days.

How our tools supported our conclusion

Our heuristics identified a number of regions that have high time complexity and/or generate output that is sent across the network. We arrived at our conclusion through manual browsing of the decompiled code in our CodeSonar/IST visualization, with a focus on the suspicious code fragments identified by the heuristics."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. GrammaTech, like Vanderbilt, noted that the problem depends on the ability to predict the random seed. GrammaTech concluded that this was not possible while Vanderbilt concluded that it was.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Our original justification discussed the possibility of breaking the PRNG; in the collaborative discussion, it emerged we were the only team to consider this in depth, although another team thought about it. During the discussion, we learned that if the PRNG could be broken, the subsequent attack would be easier than we realized. However, we also determined that breaking the PRNG is hard and would entail additional difficulties beyond the ones we foresaw during the take-home engagement. Specifically, we learned that if the PRNG could be broken, we can improve our card guess using the compressed table image png instead of the string description of the cards. This is more precise (even accounting for some room, there is at most 5 collisions for any pair of hole cards, thus well within the budget) and only requires some fingerprinting ahead of time for each card in each skin. On the other hand, breaking the PRNG is harder than we realized. In our reasoning during the take-home engagement, we forgot that each player has its own thread, so the number generators are not the same and we cannot use our messages to get the actual value of the padding. Nevertheless, at the end of a hand, the opponent hole cards are revealed so we can infer what the opponent tables were during the hand, which gives us enough values of the padding to break the PRNG. Still, breaking the PRNG naively would take years, so the attack is not feasible."

Live-Collaborative Evaluation:

Through collaboration GrammaTech confirmed their initial conclusion that the PRNG seed could not be predicted.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.7.2 Question 031 (SC Time, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"The code that is executed during a poker game before the blind bets is almost exclusively straight-line code with no conditionals that depend on the value of the secret. The conditionals only test public information: money available to both players, money in the pot, all-in situation. The only suspicious code lies in the rendering, which uses the values of the cards to fetch the right images.

Statements on if and how the observables relate to the secret

Apart from hand evaluation (which is only triggered at the end of a hand, too late for us), only rendering accesses card values through the method to_string. In particular, drawTable and reskinCard do that. The method drawTable is only given the information that is available to the player, in particular the list of opponent hole cards is empty, except at the end of the game. Card reskinning accesses the value of the cards which must be reskinned and updates the mapping in the skinnedCardsMap. […]

Statements on any potential side channels within the application

The method changeSkin clears the map skinnedCardsMap where the skin of all cards are stored. […] Therefore, after such a call, all cards must be reskinned. As this is done in an on-demand fashion, only the cards that are actually used in the current poker hand are reskinned. As the renderer is shared by both players, the time required to reskin a card may leak information on whether the card was already reskinned by the opponent, and thus is part of his hole cards.

Statements on why the application does not contain a side channel

As we can see from the code below, in the reskinCard method, the costly operation of card reskinning (which involves image manipulation) is performed on all cards (first line of the for loop). Then the mapping within skinnedCardsMap of the actual card that should be reskinned, is modified, which is only a pointer copy. Therefore, the position of the card in the deck or whether it has already been reskinned do not change the execution time of the method. […]

How our tools supported our conclusion

Our heuristics identified a number of regions that have high time complexity and/or generate output that is sent across the network. We arrived at our conclusion through manual browsing of the decompiled code in our CodeSonar/IST visualization, with a focus on the suspicious code fragments identified by the heuristics."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. There is a time side channel that allows an attacker to differentiate dealt vs un-dealt cards following a change of the card skin. An attacker can segment the deck in half, playing enough games to ensure that half the cards have been dealt under the new skin and half have not. The attacker can then use the time side channel in a new game with the benign user to determine which segment the benign user's card falls in. Oracle queries can be used to resolve the remaining entropy of the secret.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After discussing with other teams, the suspicious code still seems to be the card reskinning, given that skins are the same for both players and that the server caches them. However, even though there is some empirical variation (observed by most other teams), it is not clear that there is a good correlation as the magnitude is not high enough. Consequently, we choose to remain with our answer of "No" from the take-home engagement."

Live-Collaborative Evaluation:

GrammaTech flagged the intended vulnerability through collaboration but did not adequately analyze the strength to answer "Yes"

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.3.2.8 Searchable Blog

A.5.3.2.8.1 Question 018 (AC Space, Intended Not Vulnerable, Take-home: No)

Take-Home Response:

"Statements on any potentially vulnerable code structure in the application

The application codebase is relatively small. Nevertheless, we identified some potentially expensive memory allocations, in particular allocations of Array2DRowRealMatrix(n,n), where n is the number of blog pages.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures

There are allocations of Array2DRowRealMatrix(n,n) in the two methods named adjacencyMatrix in theParsingRoutines class. However, they are only called in the

RankingService() constructor, with n being the number of files in the directory /data/blogs/. In practice, there are 730 files in /data/blogs when the server starts. A more interesting allocation of an Array2DRowRealMatrix() is in the concatenateColumn method in MatrixRoutines.java. This method is executed from the RankingService.performRanking() method, which is itself executed each time a user uploads a new blog page using a POST request to the /submit page. However, concatenateColumn is always executed with the arguments (this.A, v) in performRanking. this.A always has size 730x730 and v always has size 730 (730 being the number of files), so the allocation of the new Array2DRowRealMatrix that would increase the dimensions of the matrix by 1, is actually never executed. The matrix does not grow in size because the attacker cannot upload many blog pages in order to increase the number of files in the /data/blogs directory. The attacker can only upload/update the page /data/blogs/userblog.html.

Statements on why the application does not contain a complexity vulnerability (strength and reachability)

We believe that, because the number of files in the server cannot be increased and is initially 730 files, there is no way to allocate matrices or vectors in the PageRank implementation that would exceed the budget of 2GB.

How our tools supported this conclusion

Because of the use of the Spring framework, WALA was not able to produce an acceptable call graph for this program. Consequently, we were not able to apply our automated analyses, and we proceeded through manual inspection of the decompiled source code."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.3.2.8.2 Question 027 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We identified a code fragment that would potentially contain a ACV-T: the method MatrixRoutines.estimateStationaryVector has a loop that computes matrix operations until convergence. Moreover, this code is executed from the synchronized method RankingService.performRanking(). The code is as follows - it is an implementation of the power iteration method to compute PageRank: […] If there is a way for the attacker to execute the performRanking method and force its execution time to exceed the budget, it would create an ACV-T for another user that uploads a file, because the execution of the second performRanking method would be blocked until the first one finishes.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures

For some values of M and v in the method above that would cause the convergence to be very slow, it might be possible that the execution time of the method exceeds the time budget. However, because of the fact that the number of pages in the blog is constantly equal to 730, the estimateStationaryVector method is always called with b assigned to the vector defined in the file data/init.dat (in RankingService.performRanking). Consequently, the attacker has no control of the b vector. However, the attacker has some control of the matrix M : by updating the blog

page userblog.html, he or she can change the values of the last column of M, which depends on what links are present in the userblog.html page.

Statements on why the application does not contain a complexity vulnerability (strength and reachability)

We have tried updating the userblog.html page with different content, but the execution time was still orders of magnitude less than the budget. More generally, if there exists some value of the Matrix A such that the power method does not converge or converges very slowly when starting from the initial vector from data/init.dat, the attacker is extremely unlikely to find it, because the attacker does not know the content of data/init.dat. Consequently, we believe that the answer to the question is no.

How our tools supported this conclusion

Because of the use of the Spring framework, WALA was not able to produce an acceptable call graph for this program. Consequently, we were not able to apply our automated analyses, and we proceeded through manual inspection of the decompiled source code."

Take-Home Evaluation:

The challenge program contains an intended vulnerability in the ranking algorithm.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"There is a way to exceed the budget by modifying the value of the matrix M in the pageRank algorithm. The attacker can control the last column of this matrix by submitting a new userblog.html page. In order for the attack to work, the last column of the matrix should be full of zeros, except the last line (that corresponds to links to userblog.html itself) that should be one. […]"

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.9 STAC SQL

A.5.3.2.9.1 Question 013 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Code that contains the vulnerability

This program implements a simple database which attempts to compress the data saved to disk using Huffman encoding, which allows the database to use less bits to store frequent data bytes. When given data to insert into field in the database the application calculates the frequency of the bytes to be inserted into that field and then calculates a valid Huffman encoding for that field. However, if the user updates a field, which has the type BLOB, the application calls HuffmanCoder.updateTree. This HuffmanCoder.updateTree can be exploited by the user to create an encoding that is very bad for a chosen data byte. It is important to note that if a field is updated and that field has any type other than BLOB the application creates a new Huffman encoding for the resulting data and thus would not represent the same kind of vulnerability. The

code that makes the distinction on the data type of a field is SQLField.setData. Also, this application is prone to write a large file because the application instantiates the entire database to one file named "db.stacsql". This happens when a SQLDatabase is instantiated with the string literal "db.stacsql" in SQLInputWorker.

Why the code contains a vulnerability

The code contains a vulnerability because an attacker can get the application to set the worst possible encoding for an arbitrary data byte. The attacker can then just update a field with a lot of the data bytes which have bad encodings and cause a blow up in size.

Why the vulnerability is exploitable

The actual attack has the following steps:

1) The attacker creates a table with one col with the datatype BLOB. This means any updates to that col will call HuffmanCoder.updateTree.
2) The attacker inserts X rows into the table with a chosen data byte to exploit. What this will do is create X fields each with the same Huffman encoding. At this point the encoding is just one bit to represent the chosen data byte. This is fine since this is the only byte the field has ever seen.
3) The attacker then makes 255 updates to the table by setting the column equal to all other data bytes aside from the chosen byte. This has the effect of updating the Huffman encoding for every row of the table. Furthermore, every update causes the chosen data byte to require one more bit to store compared with the previous update. Thus after this step all the rows in the table will require 256 bits to store the chosen data byte.
4) The attacker then updates the table again by setting the col equal to a value which is the chosen data byte repeated Y times. This has the effect of storing that value for all rows using the worst possible encoding.

Since the 8 bits of the chosen byte are stored as 256 bits on the server, the size of the resulting file written to disk after this attack is approximately (256/8)*X*Y. All requests are sent to the server as strings, and SQLInputWorker truncates all requests which are longer than 1000 characters. The update to the server in step 4 has the format

UPDATE (tableName) SET (colName)=(Y*chosenByte)

This request is limited to 1000 characters. Thus, if the chosen table name and column name are each one character long, Y is effectively limited to 735.

The question then is what is the minimum value for X for which (256/8)*X*Y >= 2MB, which defines a successful attack for this question. With Y=735, the minimal X value that satisfies the inequality is 90. However, due to miscellaneous data that is written to disk, we empirically found the minimal X to be 88 to cause a large enough file to be written to the server. With these parameters, the size of the requests given to the client script was 9448 bytes, which is under budget.

How our tools supported discovery of the vulnerability

The vulnerability was found through manual inspection of the decompiled code."

Take-Home Evaluation:

GrammaTech identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.10 Stegosaurus

A.5.3.2.10.1 Question 003 (SC Time, Unintended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret, as well as any potential side channels within the application

There is only a small number of methods in the code that actually use the secret message: Stegger.hide , Stegger.prepare_hide, NoiseAdder.addNoise, Stegger.find. Stegger.prepare_hide only assigns the message to a local variable, adds some noise to the input image (using the method NoiseAdder.addNoise) and creates a blank image for the output.
NoiseAdder.addNoise takes an input image and adds two kinds of noise to it: a uniform random noise and a Gaussian noise based on the secret message. Both hide and find perform the same essential action: walking through the input image in order to locate target bits where the secret is to be encoded / to be read. The initial index is the first pixel of the image. The subsequent locations are computed using the current secret bit written/read and a key. Given one index offset, the next one is (offset + pKey + v^80) mod image_size, where pKey is the key used for encoding/decoding and v is 3 times the value of the last bit of the message considered (either to write for hide or read for find). The secret message is written on some lower bits of the image; the difficulty for the attacker is to find out which ones. We discuss the logic involved step by step and discuss potential side channels as we go.

Random Noise

The random noise added by addNoise uses methodrandomNoise. […] As we can see from this code excerpt, the locations at which random booleans are written are always a multiple of 3. Thus, only one color (RGB format) of every pixel is modified whereas any color can be used to store a bit of the secret. Thus, two-thirds of the image's lower bits are not properly randomized.

Gaussian Noise

The Gaussian noise added by addNoise uses a standard normal distribution to write either 0 or a random bit of the secret at locations that are a multiple of the value of the key used for encoding (modulo the image size). The parameter chosen for the threshold for writing a bit is unusually large: 90.0, instead of a more typical value such as 90%. As a consequence, the probability of writing a random bit of the secret is extremely small, so small that floating point rounding may make it entirely impossible. We can therefore assume that only 0 bits are written. […] Thus, the method writes 0 at regular intervals within the image. The size of the intervals is exactly the value of the encoding key.

Computation of the Next Index

Inside hide and find, the critical operation is the computation of the next index where to write/read a bit of the secret. Both of those in exactly the same way. The only difference is in hide where the modulus computation is preceded by iterated subtractions of multiple of the modulus: […] instead of just the second line. Given a current index offset, the new index is offset + pKey + (3 * b)^80, where b is the current bit of the secret (at index offset) and pKey is the encoding key. The code computing the next index is straightline, and the only operation that may take a variable amount of time depending on the value of the secret is the exponentiation, which uses the java library method BigInteger.pow. Indeed, pow is called only with exponent 80 and base 3 or 0. In the case where the base is 0, a fast path in the implementation avoids going through the repeated squaring multiplication algorithm, which can be observed.

Execution Time

The loops of hide and find take exactly 8 * message.length() + 8 iterations (explicitly for hide, or through reading a null byte at the end of the message for find). Since each iteration is costly (it contains a modular exponentiation over a bigInteger), and the victim and attacker are assumed to be the only clients interacting with the server, this is a SCL-T that leaks the length of the secret.

Statements on why the application does not contain a side channel (strength and reachability).

- The execution time SCL only leaks the length of the secret, not its content.
- The pow side channel cannot be used on small inputs, so the information retrieved is too general to be useful. Since we do not know the length of the secret, a longer encoding time could be due to either more bits having the value 1 or a longer secret message. Even in the ideal case where the execution time SCL gives us the secret length, this would only allow us to determine the total number of bits that have value 1 in the message.
- There is no obvious way to exploit the noise side channels if we do not know the starting image.

How our tools supported this conclusion

Our heuristics identified a number of regions that have high time complexity and/or generate output that is sent across the network. We arrived at our conclusion through manual browsing of the decompiled code in our CodeSonar/IST visualization, with a focus on the suspicious code fragments identified by the heuristics."

Take-Home Evaluation:

The challenge program contains an intended vulnerability due to a race condition; however, this was discovered to be insufficiently strong.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"After the collaborative discussion, no convincing attack emerged. Therefore, our answer remains "No," as in the take-home engagement. Below is a list of interesting experimental observations gathered by other teams. As the image size decreases (the bounds for image size checks are broken), the processing time of each bit of the message increases, and the difference between 0 and 1 becomes clearly visible. This is especially true for 8x8 images but already observable for 64x64. However, this does not help at all for a worst-case analysis where we

cannot choose the image, as the difference is too small and cannot be observed accurately over network noise. The execution time of the while loop in hide is observable: for each power of 2, we get a new plateau (one more iteration in the loop). This is a mitigation mechanism that creates plateaus depending on the key size (the threshold is log_2(2^90 x image size) where 2^90 is the perf constant) and thus makes finer differences unobservable. It may also be possible to obtain the message directly during the encryption by stopping it. However, this is not a side channel in time."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.11 StuffTracker

A.5.3.2.11.1 Question 030 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Code that contains vulnerability

There is a vulnerability in the method com.ainfosec.StuffTracker.parse.SDLParser.parseStuff(). The relevant source code is the following: […] This method actually parses the input, which is controlled by the attacker, using a SAXParser. This parser is vulnerable to the billion laughs attack.

Why the vulnerability is exploitable

The attacker can create its own XML file, attack.xml, and upload it to the server. This file has the following contents and will generate a string of very large size. Note that the XML file is only 463B in size, which is much smaller than the 30KB budget for the maximum sum of the PDU sizes. […]"

Take-Home Evaluation:

The challenge contains an intended vulnerability in the SDL parsing scheme. GrammaTech appears to have identified an out-of-scope vulnerability to a billion-lols-style attack in the SAX parser.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.12 Tawa FS

A.5.3.2.12.1 Question 012 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“We believe (with low confidence) that an algorithmic-complexity vulnerability in time is not present.

Statements on any potentially vulnerable code structure in the application

Our investigations mainly focused on the Filesystem.defragment and FileTable.backupInode methods. We also investigated several other methods that were flagged by our heuristics as having potentially high time complexity, but we did not discover a way to exploit those methods as required by this challenge problem.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures

We found that we were able to cause Filetable.backupInode to enter an infinite loop, as documented in our answer to Question 20. The web server uses a pool of two threads; if both threads can be placed into infinite loops, then a benign query will not be answered within the time limit. However, when we constructed an exploit script, we found that the exploit succeeded with much less than 99% probability.

Statements on why the application does not contain a complexity vulnerability (strength and reachability)

The potential exploit that we investigated depends on the interaction of concurrent threads, and therefore is sensitive to the precise interleaving of actions taken by those threads. The resulting attack does not meet the threshold of success probability specified in the question.

How our tools supported this conclusion

The methods Filesystem.defragment and FileTable.backupInode were identified by our heuristics as having potentially large output size and high time complexity, and also as having relatively large cyclomatic complexity. We used our system's call-graph browser GUI to assist in manual inspection of the code. We manually constructed an exploit script.”

Take-Home Evaluation:

The challenge contains an intended vulnerability in the defragmentation algorithm. An attacker can cause self-recursion. GrammaTech identified the AC Space vulnerability but noted correctly that while it satisfied the lower probability for success for Question 020 it does not satisfy the 99% bound for Question 012.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“This problem does have an algorithmic complexity vulnerability in time, which is triggerable by an attacker and is within budget. The basic idea is to create a cycle in the file system structure. The attacker can then busy up a thread by getting FileSystem.findAllfiles to be called, with a defragmentation request. […]”

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.3.2.12.2 Question 020 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The parts of the code most relevant to the vulnerability are in com.bbn.FileTable.backupInode and com.bbn.Filesystem.defragment.

Why the code contains a vulnerability

The FileTable.backupInode method contains a potential infinite loop. […] The code is intended to repeatedly try to copy a block of data from one location to another until the block is copied correctly. However, if the block is read before entering the loop, and then written to by another thread before entering the backupInode loop, then all subsequent correctness checks will fail, and the loop will continue running indefinitely, increasing the length of the disk image file at every iteration. Eventually, the disk image will exceed the 1.5GB file size limit specified in the problem. Although the code contains an attempt-counter variable, that variable is never used to limit the number of iterations of the loop. Also, this part of the code does not contain any synchronization methods that would prevent concurrency bugs.

Why the vulnerability is exploitable

The attacker can cause the disk image to become fragmented by uploading a file and then uploading another file with different contents but the same filename. Specifically, a sequence of requests of the following form can be used to cause fragmentation of the disk image, as a first step of the attack:

1) upload a file with filename "a" and having length 1 byte
2) upload a file with filename "b" and having length 1 byte
3) upload a file with filename "a" and having length 4096 bytes
4) upload a file with filename "b" and having length 4096 bytes
5) upload a file with filename "a" and having length 8191 bytes
6) upload a file with filename "b" and having length 8191 bytes
7) And so on

The attacker can then request several concurrent defragmentation operations, and these have a high probability of reading from and writing to the same blocks of the disk image. As a result, these defragmentation operations will often put Filetable.backupInode into an infinite loop.

We wrote a script to demonstrate the feasibility of the attack. Our script first uploads 8 copies of a file named "a", of increasing size, interleaved with 8 copies of the file named "b", to create fragmentation. Then, our script requests 10 concurrent defragmentation operations. Our experiments with this script suggest that the attack is successful more than 50% of the time.

How our tools supported discovery of the vulnerability

The methods Filesystem.defragment and FileTable.backupInode were identified by our heuristics as having potentially large output size and high time complexity, and also as having relatively large cyclomatic complexity. We used our system's call-graph browser GUI to assist in manual inspection of the code. We manually constructed an exploit script."

Take-Home Evaluation:

GrammaTech identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"The Iowa team explained an alternative problem with Filetable.backupInode. They noticed that backupInode can get into an infinite loop if an exception is thrown inside BlockDevice.read. Upon further examination, we realized that backupInode was written with the incorrect assumption that the disk image file length will always be a multiple of 4096. In particular, if backupInode is run while the disk image file is not a multiple of 4096, the variable backupRegion will not be a multiple of 4096, and thereafter all blocks written by backupInode will fail to be aligned to 4096-byte boundaries. Then, when these blocks are later read (using Filetable.slowEquals), an exception will be thrown due to the mis-aligned blocks not having correct hash values. This exception, in turn, will prevent backupInode from ever exiting its while loop. In summary, when two instances of backupInode run concurrently, the instance that starts first is likely to change the disk image length so that it is not a multiple of 4096. This can cause the second instance to enter the infinite loop."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4 Iowa State University

A.5.4.1 Iowa State Overview

Iowa answered 26 questions in the take-home engagement with a 62% accuracy. In the take-home engagement the team had the most difficulty with SC Space and AC Time with accuracies of 44% and 57% respectively. In Iowa's take-home responses we (the EL) recognized 3 cases where the more manual components of Iowa's approach led to incorrect responses. On BattleBoats Question 16, the team failed to recognize that the attacker had direct control of a loop's termination condition. On BattleBoats Question 21, the team correctly observed the use of a hash map in the looping structure and that the loop terminates on sufficient convergence; however, they missed the condition to trigger the slow convergence. Finally, on Poker Question 004 in their analysis Iowa appears to have mistaken the method used to format the output string as a method used to add random padding; as a result, the team missed where the random padding occurred.

Poker Question 004 presented an interesting collaboration opportunity as GrammaTech and Colorado identified a weak side channel and ruled correctly that it was not sufficiently strong. Iowa and Vanderbilt identified potential side channels but ruled them sufficiently strong. The decision point for the presence or absence of a vulnerability was due to whether the random seed used to pad packets could be determined. As the result of collaborations, all teams correctly concluded that there was no way to determine the random seed for the benign user and predict the padding sequences, as separate *Random* objects are used in the benign user and attack threads and the challenge question limits the attacker to a single game with the benign user. This was a highlight of what was expected from the collaborative engagement.

In the live collaborative engagement, Iowa answered 27 questions with the highest overall accuracy of 100%. The team's 38 percentage-point increase from the take-home to the live engagement was the third highest. We hypothesize that the team's understanding of the functions

of the challenge programs even in cases where they answered incorrectly during the take-home component may have been a factor in their live-collaborative engagement performance.

A.5.4.1.1 Take-Home Engagement Scores

Table A-93: Engagement 5 ISU Take Home Engagement Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	4	44	4	44
SC Time	5	4	80	4	80
SC Space/ Time	0	0	0	0	0
AC in Space	7	4	57	4	57
AC in Time	5	4	80	4	80
Total	**26**	**16**	**62**	**16**	**62**

Table A-94: Engagement 5 ISU Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	20	13	**65**	13	**65**
Not Vulnerable	6	3	**50**	3	**50**
Yes Answer	16	13	**81**	13	**81**
No Answer	10	3	**30**	3	**30**

A.5.4.1.2 Live Engagement Scores

Table A-95: Engagement 5 ISU Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	9	100
SC Time	5	5	100
SC Space/ Time	0	0	0
AC in Space	7	7	100
AC in Time	6	6	100
Total	**27**	**27**	**100**

Table A-96: Engagement 5 ISU Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	21	21	**100**
Not Vulnerable	6	6	**100**
Yes Answer	21	21	**100**
No Answer	6	6	**100**

A.5.4.2 Iowa State Responses

A.5.4.2.1 Accounting Wizard

A.5.4.2.1.1 Question 011 (SC Time, Unintended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Given that an employee knows a project exists, they can use the update hours functionality to reveal the project expenditure. On average, the response time to complete an hours update that is over budget will complete faster than an hours update that is within the expenditure limit. […]" (Note, the team's response contains much more detail on the identified vulnerability and performed analysis; however, much of this overlaps with the analysis identified by other teams and so was left out of this short extract of the response.)

Take-Home Evaluation:

There was a mistake in packaging this challenge program: the application came provided with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects' contents or states.

Iowa and Northeastern identified an unintended vulnerability during the take-home where under certain conditions, an attacker-controlled input of hours is directly compared against the secret. Iowa's exploit as presented is insufficient to exploit this, but the amplification discovered during the live collaborative engagement or using the remaining input budget to amplify the side channel are both sufficient to exploit the vulnerability. As an exploit is not required and all components of the unintended side channel vulnerability were identified, the post-engagement evaluation is Correct.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Working with other teams Iowa similar to the other teams used the SC Space vulnerability to amplify the strength of the reported side channel vulnerability.

Live-Collaborative Evaluation:

See the take-home evaluation.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"This application does have an AC Space vulnerability that comes extremely close to the resource usage limit, but the input budget is too small to exceed the resource usage limit. An exploit for the weak AC Space vulnerability is included with this report. […] An analysis of the login functionality reveals no obvious way to escalate permissions of a role within the application within the given budget. Each endpoint URLs is validated against a list of hardcoded user assigned roles. If an endpoint is not authenticated with the proper role, the "before" handler throws a 401 (unauthenticated) exception preventing execution of the role restricted handler logic. The following table reflects the application enforced endpoint restrictions. […] After reducing the search space of the application to attacker-controlled code, we search for any File I/O applications that amplify data sizes. Particularly, we are looking for approximately a 20x amplification in terms of the original attack input budget. Examining the application's interactions with File I/O APIs revealed several direct file interactions in the FileStore and BaseObject classes, however neither of these were reachable from the attacker-controlled endpoints after the previous code elimination was performed. All remaining File I/O affects the .accounting.log file via the debug, error, trace, info, and warn methods in com.bbn.accounting.logging.Logger. The content of the log message depends on the callsite, the current language setting (using the /lang/:tag endpoint), and whether or not an exception is included in the varags Object parameter. Of the Logger callsites controllable by the attacker that have not already been eliminated one callsite stands out in the SiteWideRoutes.exception handler, which catches exceptions of type APIException. […]"

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attempt to switch the local to and unsupported on triggers verbose logging mode that can be used to submit inputs that cause the server to exceed the resource usage limit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Based on the E5 White Team application description and question for the take home engagement, it was unclear to us whether a project could exist at runtime given the application scenario. However during the collaborative engagement, a clarification was made that "You can assume that there are active projects that the user can charge time to", which changes this answer. For the first interpretation of the question (when no projects exist) the analysis in our original report for the take home engagement holds true. When the clarification that the attacker has access to a project at runtime is true the answer changes and the rationale is explained in the "addendum" section of this report. An exploit is included with this report.

Parts of this addendum were derived from an analysis performed by the Utah team, in particular that the log level is changed in the event of an invalid set language locale request. If it is possible that a project exists at runtime then it is possible to exceed the resource usage limit. This is possible by setting the log local using the ""/lang/:tag" endpoint declared in the SiteWideRoutes.installRoutes to a non-existent language (example: "xxxx") which causes an exception in Logger.setLocale. During the construction of the thrown LocaleNotSupportedException exception the log level is set to the TRACE level. Once the log level is set to TRACE, the BaseObject.uuid method's call to Logger.trace is enabled to write to

the log file. Using RULER's callsite analysis reveals there are two callsites to the Logger.trace method. […]"

Live-Collaborative Evaluation:

Iowa identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.1.3 Question 023 (AC Time, Intended Vulnerable, Live: Yes)

Live-Collaborative Response:

"This new report for a previously unanswered question was made possible by a collaboration with Utah. By adding new items of unique names and varying costs the report expenditure operation becomes very expensive in time. An exploit is included with this report.

RULER automatically identified the application entry points (application main method and request handlers). From this information the expenditure report "/projectmanager/report/expenditure" endpoint was located manually, and control flow was tracked using the call graph. Since according to the challenge question, the report expenditure is the action that triggers the AC Time vulnerability the vulnerability must lie in logic that is invoked in response to the report expenditure endpoint handler. Using a forward call graph from the report expenditure endpoint makes it possible to compute a restricted set of the functions that could contain the vulnerability.

The LCG is a call graph with induced on the methods containing loops with annotations (highlighting) on the methods containing loops and the methods that are called within loops. Using the LCG we can further restrict the call graph to identify the interesting intraprocedural uses of loops. The restricted LCG reveals many interprocedurally nested loops in the BudgetSolver logic.

In particular BudgetSolver.getAffordableLines calls BudgetSolver.vectorDec in a loop (and as the loop termination condition). The BudgetSolver.vectorDec method also contains a loop that monotonically decreases over the set of elements in the given vector. Utah pointed out that there is additional looping logic that was not capture in our analysis due to the use of lambdas filter and collect methods. This understanding is needed to identify the values that are necessary to cause the AC Time vulnerability. The inputs to the BudgetSolver.getAffordableLines include item costs that are less than or equal to the expenditure limit. Another use of lamdas with loops is to sort the input passed to getAffordableLines. Jointly, ISU and Utah hypothesized that the combination of these loops becomes very expensive for projects with multiple items. ISU developed an exploit POC (included with this report) that proved the AC Time vulnerability can be used to exceed the resource usage threshold.

The ISU and Utah each sent one analyst to collaborate for about half an hour before the hypothesis was proposed. Another hour was taken to develop the POC exploit. ISU plans to account for loops in lambda functions in future tooling to prevent missing vulnerabilities involving deeper understanding of lambas in the future."

Live-Collaborative Evaluation:

Iowa identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The attacker goal is to recover the number of budget items defined as "employees on the project + number of expensed items" for any project that the attacker works on. The description of this application states that "employees work on projects". While there are two other roles (managers and project managers) in this application neither of their user interactions include "working on a project". The application does not however restrict managers and project managers from logging project hours (which is the only way to "work" on a project). An analysis of the login functionality reveals that there are only 4 possible user accounts on the system, which are hardcoded and cannot be added to or mutated. There are two employee accounts, a manager account, and a project manager account. If we are to follow the description definition that only employees will log time in a normal set of interactions, then the range of employees that may work on a project is 0 to 2, but if we included managers and project managers then the range is 0 to 4. In either case the number of employees that may work on a project is negligible compared to the number of items expensed on a project. So if we are able to estimate the number of expensed items on a project then the attacker goal is easily attained with at most 3 or 5 oracle queries, assuming employees or employees and all managers roles are permitted respectively. […] When a batch of items are ordered with the "/orderItems/:projectName" endpoint, EmployeeManager.orderItems method is invoked, which retrieves the associated project and invokes FileStore.newItem. The FileStore.newItem method constructs a new com.bbn.accounting.objects.Item object, which is then serialized to the file store in the ".store" directory. The FileStore.newItem also updates the corresponding file store project file in the ".store" directory. Both files are serialized with the inherited BaseObject.write method, which invokes the BaseObjectMarshaller.toBytes method. The BaseObjectMarshaller.toBytes serializes an object by writing an object serial ID (a function of the object's hash code), the object name (in this case the item name, such as "pen"), the annotated fields (cost and quantity for items), and a list of references in the form of a SHA1 hash of the item name to the object's children (for projects this is a list of the project's expensed items, for items this list is empty). Since the FileStore.newItem updates the project file when a new item is created, we expect the project file to grow by the length of one child (the length of a SHA1 hash, 40 bytes, plus a few bytes as delimiters). Experimentally, we see that adding items grows the project file at a rate of 44 bytes per item order. This plot indicates that if an attacker is able to observe the size of the project file, there is a side channel in space that could potentially be used to reveal the number of expensed items on a given project. This is an example of the White Team's canonical example Category 16, where the secret is leaked into a different domain. In this case, the domain is the size of the project file. Note that a similar analysis of the "/updateHours/:projectName" endpoint, which leads to the EmployeeManager.updateHours method that calls FileStore.newHours reveals a similar behavior when employee hours are logged. Since employee hours are accumulated after the first hours are logged, the project file store file is updated with a reference to the accumulated employee hours. This indicates that the size of the project file also grows when employees start working on a project, but this analysis is ultimately unnecessary because the secret is the sum of the items expensed and number of employees on the project and not each value individually. Since the number of user accounts in the application are limited to two employees, oracle queries should be sufficient for recovering the number of employees working on a project's contribution to the overall secret. […]"

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.2 BattleBoats 1

A.5.4.2.2.1 Question 006 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Since this question asks for a remote DoS AC attack, the input of interest would be something that attacker can control but it also impacts the operations on the victim side. By looking at the game protocol it is clear that the input of interest can be either cannon coordinates or the parameters to a shot by cannon (height, velocity, angle of elevation etc.). Since the victim decides where attacker's cannon would be placed, parameters to a shot is the input of interest. In short, the only possible scenario for remote DoS is that attacker makes a shot (shoots his cannon) and the victim exceeds the resource usage limit while computing the damage. […] Next, we focus on the loops in this call graph. There are only 2 loops - one in calculateStrikeTime and another in updateStrikesAsSunk. The loop in OceanBoard.updateStrikesAsSunk() takes in the computed strike coordinates and updates them as the shot in an iterator-based loop. A cannon can at the maximum target 4 squares (the neighbors of a point), so it is unlikely to be a problem. Eventually, we are left with the loop in StrikeLocator.calculateStrikeTime(). It seems to compute the time of flight of the cannonball. This loop is weird, as instead of using a simple projectile motion equation, it tries to compute a series and relies on its convergence for termination. This is suspicious. We have shown the code as displayed by our CFR decompiler view and the loop termination PCG below. […] We were able to trigger this code. In fact, whenever cannon shoots this code will be triggered, so there are no special conditions to trigger this. With that in mind we investigated for the hypothesized input. We performed dynamic analysis (unit testing) on the loop to check if it ever exceeds the maximum number of iterations allowed. We observed that if the height of the cannon from which shot is made is 0, then the other parameters are irrelevant and the loop always converges slowly. But we also found out that the loop always breaks out when the iterations allowed exceed the constant upper bound. Hence, the only possibility was whether it exceeds the resource usage limit within those 25 iterations for height = 0. Our dynamic analysis revealed that it never exceeds 3 seconds for such a request."

Take-Home Evaluation:

An attacker can submit inputs that cause the projectile to never hit the ground causing the method to loop endlessly. In their response, Iowa narrows the scope of the attacker control. While not the cause of the incorrect response, the attacker can use more than the benign set of commands to trigger a vulnerability. Iowa's response indicates that they identified that the code that computes the time for the projectile to impact can be controlled by the attacker. Iowa's analysis fails to identify the conditions for triggering the vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After collaborating with Colorado and Northeastern, we learned about the possibility of causing a AC vulnerability in Time using the height parameter. The AC vulnerability they found is caused by supplying a negative height. To validate that possibility, and in turn their exploit we did the following additional analysis. The flaw in our previous argument was we assumed the guards are strong enough to prevent height values other than the range specified in the description. To convince ourselves, we explored the conditions that guard the height from being negative. In order to do that, we started from the WarShipsDispatcher.handleReceivedMessage(), which is the first place the height argument of the shoot command is processed. We tainted that variable and induced that taint on the call graph. This gave us all the method that are tainted by the height variable. The resultant taint graph had 1327 nodes and the induced call graph had 165 method nodes. Since we are interested in finding a weak guard on the height value, we decided to select the branch conditions out of the tainted nodes and examined them first. There are 22 branch conditions among the tainted portion of the app. We examined these branches manually. The relevant branch is in the constructor method StrikeLocator, which checks whether the height is negative or not. We unit tested this guard on its own first, and it does not allow negative height. Following is its source code as displayed by the decompiler view […]. We reasoned what happens after that. If the error message is set, the client throws an illegal argument exception in the method Competition.fetchStrikeSquares and it does not do anything after that."

Live-Collaborative Evaluation:

Iowa identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.2.2 Question 028 (SC Space, Intended Not Vulnerable, Take-home: Yes, Live: No)

Take-Home Response:

"The secret is the coordinates of attacker's cannon. Since this is a peer to peer application, the only communication is between peers. Hence, the observables would come from the code segments corresponding to communication between peers. We tainted the secret, i.e., parameters of the method WarShips.setCannon(). This gives 3 loops and 52 branch conditions. Out of the 3 loops, 1 was simply to induce random delays. So we are left with 2 loops and 52 branches. The 2 loops are used to encode the location coordinates. […] The IPCG above shows the evidence. The nodes with Red boxes around them are the loop headers. The other 3 nodes which are of the same color as the loop headers correspond to the operation of placing a cannon. As seen in the figure, they are called by a common ancestor. These code segments are exercised during placement of cannon. According to the game protocol, a message must be sent after the cannon is placed as an acknowledgement. This means the observable should include the size of packets in the request that acknowledges that the cannon has been placed. These two loops claim to "encode" the cannon coordinates before sending a message to the opponent signifying end of placing the ships (and cannon) stage. This encoding is the addition of blank spaces at the end of the message conveying to the opponent that the cannon has been placed. A closer inspection reveals that the encoding adds exactly (x+23y) blank spaces as padding where (x,y) are the secret coordinates (cannon coordinates). This (x,y) doesn't come from any other coordinates. (Shown below is the loop's control flow graph. These loops are nested so only one graph is provided.

The control flow node which adds a blank space is marked using a red box.) This tells us that by observing the size of this packet of the message, an attacker can figure out (x+23y) where (x,y) are the coordinates of the cannon (x and y are integers). This counts as 1 active operation.

How observables reveal the value of the secret

To work out the probability for this question, we need to consider the range of coordinates allowed. An inspection of the decompiled code where board size is set tells us that it cannot be more than 20 x 20 (in WarShips.setCompetitionParameters()). Also from Competition.assignCannon() we know it can't be less than 1. Which means 1<= (x,y) <= 20 i.e. there are 400 different combinations in the worst case. But what we also know is x+23y, which gives us the equation (x+23y) ≡ x (mod 23). As there are possible 23 remainders and x can only go from 1 to 20, we obtain x. By substituting for x in the above equation, we also obtain y."

Take-Home Evaluation:

We don't believe the challenge is susceptible to this vulnerability. Additionally, Iowa's budget calculation appears to be based on attacker inputs to their application console, not remote requests to the application instance under attack.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After collaborating with the control team, we understood the flaw in our answer. We stopped the analysis after finding the padding. For inputs we tested, we were lucky enough to find the exact cannon location. But for the worst case a more thorough analysis is required. We took one step back and tainted the message that is generated at the end of EndPlacingShipsCommand and induced that taint onto the call graph. This returned all the methods that are relevant to the message formation. The induced call graph is shown below. This reveals the existence of the method Warships.rarefy which we had missed earlier. The message goes through this method after the padding is added and the output is passed one more time to the same method. […]

In order to be thorough, we audited the other location where cannon location comes into the picture, when attacker shoots with the cannon. After analyzing it dynamically, we did not see a reliably observable timing difference. Hence, we concluded our analysis and decided to flip our answer from Yes to No."

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge does not contain an intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.3 BattleBoats 2

A.5.4.2.3.1 Question 010 (SC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"[…] In this application, there are limited actions a player can take. A player must place the cannon and boats before he shoots on the opponent's board. The only repeatable action a player can take is to take a shot. In addition, in a shot, the only thing that is related to the coordinates of the cannon (secret) is the distance it hits. If a player is able to control 4 parameters (height of cannon, shooting velocity, elevation angle, and board angle), he can virtually control the distance

corresponding to a shot. We performed experiments with various values of the parameters. We found that if the player sets height to 4 and the elevation angle to 45 degrees, the shooting distance is approximately equal to the velocity. Therefore, if the attacker takes shots along X-axis and Y-axis and controls the velocity, he will be able to make on-board and off-board shots. If there is a difference in response time between on-board and off-board shots, the attacker may be able to reveal the coordinates of the cannon by observing response time with different distances that the cannon hits. […]"

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.3.2 Question 021 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"The loop in Competition.isOver() is essentially a loop that repeatedly checks if the competition is over or not. This loop doesn't consume any memory and hence can be safely ignored. The loop in OceanPanel.updateHitsAsSunk() takes in the hit locations computed and updates them as hit in a iterator based loop. It simply maintains a HashMap of hit locations. As the maximum board size is 20x20 - this HashMap can grow at the worst to 400. Its key is coordinates - 2 integers = 8 bytes. Its value is collection of 2 strings and 1 int. One of the strings is always a character (1 byte) while the other is the name of the ship which is at max 10 characters (Battleship) = 10 bytes. Thus a gross upper bound comes out to be (8+11)*400 = 7600 bytes which is way below the resource usage limit. Eventually, we are left with the loop in StrikeLocator.calculateStrikeTime(). It seems to compute the time of flight of the cannonball. This loop is weird as instead of using a simple projectile motion equation it tries to compute a series and relies on its convergence for termination. This is suspicious. The code snippet of the loop as displayed by our CFR decompiler view is below and its loop termination PCG. […] This loop is a convergence loop which tries to compute a series which is supposed to converge. It has a weird branch condition in it. (highlighted Cyan in the graph and bold face in the code). It is checks if the number of iterations are exceeding the number of iterations allowed (hard coded to 25), in short checks if the series is converging too slowly. Interestingly enough, instead of breaking the loop in case of slow convergence, it creates a new HashMap and fills in the trajectory data. This HashMap is filled with a BigDecimal and 4 integers. In short for every iteration it adds 48 bytes to it. This will happen every time a shot is made and the loop is made to iterate more than the hard coded upper bound. More importantly this will happen on the victim's side. We hypothesize that this code is reachable and there exists an input within budget, that causes it to converge very slowly. The reason we are interested in slow convergence is because the longer this loop runs the more the HashMap grows and consumes memory. Hence, slow convergence correlates to increased memory consumption in this case. We were able to trigger

this code. In fact, whenever cannon shoots this code will be triggered, so there are no special conditions to trigger this. With that in mind we investigated for the hypothesized input. We unit tested the loop and discovered that it consumes about 7008 bytes in the worst case - when height of the cannon from which the shot is made is set to zero. Which means the game should drag on till about 100,000 shots are made. The board has a limit of 20x20, which means 400 squares. While the attacker can drag the game by shooting outside the board, it is unlikely victim would do the same. Hence, this vulnerability hinges on the improbable scenario of the game being dragged on for a long, long time. Thus, it is not a strong AC attack. Another possibility is to send several requests parallely, which we tried. The hindrance there is that we need to spawn about 100,000 threads simultaneously for the attack to succeed. This is impossible with the threadpool available on the reference platform. A final possibility is race conditions. We were not able to trigger a race condition in this app. We looked at the threading subsystem interactions to statically determine if there is such a possibility. This app maintains a scheduler to schedule the threads. This scheduler will make other threads wait and avoid race conditions."

Take-Home Evaluation:

An attacker can trigger slow convergence of the algorithm that calculates the time for the projectile to strike. This uses memory with each iteration of an internal loop. Iowa identified the potential for slow convergence resulting in a vulnerability due to the user of HashMaps. However, as the team did not identify the method for triggering slow convergence they reported that the application was not vulnerable.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After collaborating with Colorado, we learned about the possibility of causing a AC vulnerability in space by taking advantage of the precision BattleBoats allows in the value of the arguments specified.

The code which we argued has a potential to cause AC space vulnerability is indeed the cause of the AC vulnerability Colorado found. What we could not find out was the input to that code which would trigger the vulnerability. Colorado's fuzzer was able to generate an input to trigger the vulnerability. The issue occurs when you provide an input which has lots of decimal places but still within precision. Since that convergence variable depends upon the subtraction of two bigdecimals, if one provides a value with such a precision, then it takes a very long time to converge. That is the vulnerability. We were able to verify Colorado's exploit and hence we are flipping our answer to Yes."

Live-Collaborative Evaluation:

Collaborating with Colorado, Iowa identified the intended vulnerability. The difference between the take-home and the live for this team appears to be that the method for triggering slow convergence was identified during the live collaborative engagement with help from Colorado.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.4 BraidIt 1

A.5.4.2.4.1 Question 002 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“Our tool helped us investigate all places in the code where a potential side channel in time can occur where the secret (plaitNumber) can be revealed. Our tool found no such evidence. The secret here is the braid selected by the user in round 1 and 3. Our tool was able to identify the secret as the field plaitNum in SelectPlaitCommand class. The observables here are the network traffic. Assuming player A is the attacker who initiates a game, our tool was able to identify the place where packets were sent back by player B to player A after player B selected the secret. Since this is a peer to peer app, our Dashboard identified methods in the io.netty.channel package as possible application entry points or communication interfaces. We are able to identify which commands send data (potentially a secret) back over the network. The PCG below show all commands that can send data back to the network. We were also able to identify the TransmitModifiedPlaitCommmand class which sends the modified braid and the original braids back. Player B selects a braid using and then optionally modifies it, and then sends it back. The SelectPlaitCommand class handles the braid selection and the TransmitModifiedPlaitCommmand handles the sending of the braid. We used our tooling to find potential side channels with respect to these commands. Our tool helped us audit relevant reachable loops. Our tool found reachable loops (see Fig: LCG below) from TransmitModifiedPlaitCommmand.execute() but did not find any differential paths related to secret that would lead to a side channel in time. We did a taint analysis from secret (Fig 2) and did not identify and potential difference in timings that would be caused related to secret.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.4.2 Question 007 (SC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“There is a side channel, where when the benign user (player B) sends back the modified braids back to the attacker (player A) in round 1, the packet size can be used to figure out the braid that player B selected. The same is true for round 3 as well. This side channel is only successful, if the game is set to an error state before round 1. In the error state, the getPlaitString() returns a string whose length depends on the plaitNumber (the selected braid). The length for the returned string is 50 times the selected braid number. This braid is returned along with all the 5 braids, and since we know the original lengths of the braids, the difference in length of this total packet can reveal the braid that is selected. […] The TransmitModifiedPlaitCommmand class uses a builder to build the modified String and all the 5 braids. The first part is the selected braid which is obtained from the getPlaitString command. Our backward Taint Analysis revealed that the length of this String might depend on the selected plaitNumber. The taint on plaitNum in SelectPlaitCommand class revealed a differential branch point where in one branch(branch1), the plaitNumber was set to the secret(the selected braid_number) in one PlaitSelectedPhase instance,

and in other branch(branch2), that was always set to 1. The method getPlaitString() in the PlaitSelectedPhase class returned the String corresponding to the selected Plait. The length of the String returned is always 50 times the plaitNumber. So in the case, where branch1 was executed above, the plaitNum will be set to the selected plaitNumber (1, 2, 3, 4 or 5), and the length of String returned by getPlaitString will be 50, 100, 150, 200 or 250 respectively. Here is the full taint graph, which shows how the plaitNum in SelectPlaitCommand flows to the TransimitModifiedPlaitCommand. Continuing our analysis, we can see that this can only occur if errorPhase variable is not null, which occurs when there is an error in validating the length of a braid. […]"

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.4.2.4.3 Question 017 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"There is an algorithmic complexity vulnerability that causes the challenge program to save multiple files that exceed the resource usage limit. An attacker using a malicious client can send modified braids that cause the victim (player B) to run an infinite loop while logging status message to a file. These log messages add up to exceed the resource usage limit. […]"

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.5 BraidIt 2

A.5.4.2.5.1 Question 001 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"The observables here are the network traffic. Our tool was able to identify the place where packets were sent back by player B to player A after player B selected the secret. Since this is a peer to peer app, Our tooling was to identify methods in the io.netty.Channel package as possible application entry points or communication interfaces. We were able to identify the Dispatcher and WeaveItDispatcher as a dispatcher class and from there the SelectWeave command class, the execute method of which is executed, when a weave (secret) is selected. We were also able to identify the TransferAlteredWeaveCommmand class which send the modified braid and the original braids back. The TransferAlteredWeaveCommmand class uses a builder to build the modified String and all the 5 braids. The first part is the selected braid which is obtained from the getPlaitString command. Our backward Taint Analysis revealed that the length of this String

might depend on the selected weaveNumber. The taint on WeaveNum in SelectWeaveCommand class revealed a differential branch point where in one branch(branch1), the weaveCount was set to the secret(the selected braid_number) in one WeaveElectedState instance, and in other branch(branch2), that was always set to 1. The method grabWeaveString() in the WeaveElectedState class returned the String corresponding to the selected Plait. The length of the String returned is always weaveCount times the length of the selected braid. So in the case, where branch1 was executed above, the weaveCount will be set to the selected weaveNumber (1, 2, 3, 4 or 5), and the length of String returned by getPlaitString will be x, 2x, 3x, 4x or 5x respectively, where x is the length of the selected weave. So there is definitely a potential side channel if this can be triggered. Following the taint backwards and using our program dependency analysis smart view, we concluded that the weaveNumber is only set in certain case, and that is dependent on result of SelectionState.takeErrorState(), and that it is only set when SelectionState.takeErrorState() != null. Continuing our analysis, we can see that this can only occur if errorState variable is not null. Backward taint analysis reveals that this is always the case. The errorState variable is initialized to empty string (not null) and if an error occurs it is set to an error String. We conclude that there is a side channel which is always triggered. The size of the packets sent by the TransferAlteredWeaveCommmand is affected by the selected weave Number. […]"

Take-Home Evaluation:

The challenge program contains several vulnerabilities, ranging from reducing the space of the secret to observing leakage in output packet sequences. Iowa identified a vulnerability but incorrectly ruled it of insufficient strength.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After collaboration with Colorado and GrammaTech team, we are flipping our answer to Yes. We had already found the side channel but had concluded earlier it was not strong enough. As written in our explanation earlier we had said that the lengths of the selected braids can be changed by the user.

However, if the attacker sets all the lengths to say 52 (max length), then the victim cannot change the lengths any more. In this case, we know the size (mb) in the equation will always be 52 and we can find "n" the secret.

$$S = O + mb * n + L_1 + L_2 + L_3 + L_4 + L_5$$

We were under the impression that the attack has to work in the worst case, and assumed the attack has to work for any "sets" of lengths. However, this seems not to be the case and we change our answer to Yes given this information."

Live-Collaborative Evaluation:

Iowa identified an SC Space vulnerability in fixing the lengths and observing the resulting response size to leak the secret. Collaboration allowed Iowa to determine the strength of the vulnerability identified during take-home.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.5.2 Question 022 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We hypothesized that the code executed on one of these ExecutorServices Threads would cause that thread to run for a long time. This would keep that thread from responding or executing the 'help' command. Looking at the construction of these 2 ExecutorServices (Dispatcher.scheduledExecutorService and Dispatcher.displayExecutorService), we see that both of them use only a single thread. This means that we only need to find a single place in the challenge program that would run for a long time to block all of the threads of one of the ExecutorServices. We looked at both ExecutorServices and found that the field Dispatcher.scheduledExecutorService is used in more places and so most likely has more candidates for blocking the Thread. We again used the Atlas Call script (see image below) to find all methods called by each of the tasks run on the scheduledExecutorService thread. Using this, we found that the class NewConnectionManager calls WeaveItDispatcher.handleNewConnection, which calls WeaveIt.addConnectionManager, which calls Channel.close().sync() from CommunicationsConnection.close. Since this appears to be performing what could be a long operation on another Thread and then waiting for the result, we decided to test how long this operation takes. In order to trigger this call trace, it is necessary to attempt to connect another player to a user that is already connected to a player. Doing this we found that the benign user's client appears to not respond to any command given after this happens (including the 'help' command). This satisfies the requirement of this question that it should take at least 300 seconds (measured from the time the benign user enters a 'help' command to the time when the application outputs the response to the console). Another AC Time vulnerability satisfying the requirement of this question was found in braidit_2. However, it does not relate to the 'help' command. This vulnerability is the same as the vulnerability found in braidit_1 for Question 17 which puts the client into an infinite loop. For braidit_2 however, the malicious modified braid Strings is "AbcdefghijklmnopqrstuvwxyzAAAAAAAAAAAAAAAAAAAAAAAAAA" and each of the 5 original braid Strings are "" (empty strings)."

Take-Home Evaluation:

The first vulnerability is one that we believe only results in a self-DoS. The second vulnerability is an unintended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.6 IBASys

*A.5.4.2.6.1 Question 015 (SC Space, Intended Vulnerable, Take-home: No, Live: *Yes)*

Take-Home Response:

"[…] The valid tokens for the users (returned to the client on successful authentication) is also loaded in this function. However, since the question relates to impersonating the user in the ibasys application (not impersonating the user on a third-party application using the token from ibasys) as per the app description, we understand that the secret is the pcode and not the returned token. The rest of the analysis and report is based on this understanding. The 10000-byte chunks of the encrypted passcode image sent in the client requests are received by the server in a loop inside the IBASysServer.init entry point. Tracking the control and data flows from the point of reception in IBASysServer.init using our smart views, we found that parts of the image received from individual client request packets are decrypted and appended to the LoginSession fields databuff and tsadatabuff. Taint analysis revealed that eventually, the app saves image contents in the field ImageMatcherWorker.imagedata, from where it is accessed for comparison with the secret. […] We hypothesize that the loop L2 in ImageMatcherWorker.run thread that compares a byte array corresponding to the secret (UserDatabase.pcode loaded during server initialization) to the client input image in a loop (step 5 above) is a potential side channel, because the result of this comparison is used to send back either the valid token or a randomly generated (invalid) token back to the client (decompiled code snippet below). […]"

Take-Home Evaluation:

Iowa's response begins with an incorrect assumption that leaking the token returned by the application does not represent impersonating the user. The challenge program accepts authentication requests and upon valid authentication returns this token. Impersonating the user means being able to get all responses from the application that only that authenticated user can receive. This application only returns one response to an authenticated user, a token. One way to obtain this token is to determine the user's passcode image/string. However, if an attacker can find another way to perform the only action that a benign user can do, get the application to return a token for that authenticated user, then this is sufficient. The confusion in this instance understandable arose as this returned code is for another (non-included) application. The challenge contains an intended vulnerability that allows the attacker to via a race condition leak the secret bit-by-bit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We stick to our "No" answer. We collaborated with Control team, GrammaTech and Northeastern/UCSB/CMU.

In addition to the hypotheses we already described in our report, one additional hypothesis came up during our discussions, seemed to indicate a trivial side channel which can be exploited as follows: The attacker can send his passcode image to the server such that the first packet contains his username, but the rest contain the username of the victim. The code that returns the token only looks up the token for the username from the last packets. As a result, the token the attacker actually receives is the token of the victim.

We however reject this as a valid exploit because the attacker does not "learn information that allows him/her to impersonate the target user" as the question requires. Specifically, the three pieces of information, namely the target username, attacker's username and attacker passcode image needed for this exploit are all already known to the attacker. Therefore, we do not think this is a side channel in space in the IBASys application. We have also noted this at the beginning of our original report for the take home engagement. Our interpretation of this phrase

in the question is that the attacker should find a valid passcode image input, which is successfully authenticated by the IBASys application for the target username."

Live-Collaborative Evaluation:

Impersonation of another user is to span the set of possible outputs from the server for all authenticated user requests. This application's only output for all authenticated user requests (only request is a username + passcode image) is to produce a (token) for that authenticated user. Therefore, as this attack accesses the only output that an authenticated user can receive, this response is changed to an out-of-scope "Yes" and given a post-engagement ruling of correct.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.6.2 Question 024 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"[…] Audit of the loop in Scalr.scaleImageIncrementally: We found that Scalr.scaleImageIncrementally is not reachable in the call graph from the entry point. Thus this loop cannot cause a timing side channel. Audit of the loop in Scalr.apply: This loop does not iterate over the individual bytes of the image sent in the client request, nor it is compared to the secret (value in the map UserDatabase.pcode for the given username). Hence we conclude this loop cannot contribute to a timing side channel. […] Loop L1 cannot cause a timing side channel because it is not comparing the image bytes from the client requests against the secret. We hypothesize that the loop L2 in ImageMatcherWorker.run thread that compares a byte array corresponding to the secret (UserDatabase.pcode loaded during server initialization) to the client input image in a loop (step 5 above) is a potential side channel, because the result of this comparison is used to send back either the valid token or a randomly generated (invalid) token back to the client. Hypothesis 1: The loop L2 in thread ImageMatcherWorker.run whose termination depends on a comparison between parts of the client input and secret induces a timing difference. Refutation: We performed dynamic analysis by instrumenting the loop L2 with timer and iteration counter. We also extracted the code for this loop (shown below) and unit tested it. We then tested this setup with mutiple client requests obtained by modifying various sets of bytes in the correct image. […] The ImageMatcherWorker.success variable is the variable that determines whether the client is authenticated or not (it is set in loop L2). So any differential behavior must be predicated on this value. Using our Taint smart view, we traced the use of ImageMatcherWorker.success. We identified two differential branches that depend on the value of ImageMatcherWorker.success (figure below). Both are in IBASysServer.init, in the loop that prepares and sends response to the client. One is responsible for difference in behavior of the server in preparing response so that when the correct_image is sent, the returned token is the valid token stored in the server's database for this user, and when incorrect_image (or an image that causes loop L2 to terminate early) is sent, the response contains a randomly generated token. This differential branch is found in the IBASysServer.init when preparing the response […] We performed unit testing to find that there is no significant timing difference between creating random token versus returning the valid token. […]"

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.6.3 Question 029 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"[…] Log file is written only if a specific System property is set. The app never sets the system property programmatically nor it is set as an option when starting the server with the given script. Hence, we reject the hypothesis that Sclar.log causes an AC Space vulnerability […] We reject hypothesis that there is File I/O in the app that can cause AC Space. […] Files or logs are written when exceptions are thrown and caught. These logs combine to exceed the resource usage limit […]"

Take-Home Evaluation:

The persistence of the session object can be used to bypass the input guards and trigger the intended vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"The teams we collaborated with (Colorado and Northeastern) informed us of exploits they found for this question within the budget. Based on the discussions, we all concurred that there are 3 pieces of information crucial to this exploit:

A. In LoginManager.parsePackerPart, an ArrayIndexOutOfBoundsException is thrown if the images sent by the client are zeroed out. This exception triggers the logging in the catch block of IBASys.init.
B. The file name saved to disk has a 1-1 correspondence with the hashcode of the username.
C. The AES key sent in the client packet has a 1-1 correspondence with the session to which image data from the packet is appended. Specifically, while adding image data from an incoming packet to the current session, the application does not validate that the username in the incoming packet equals the username in the current session.

We already had known about (A) and (B) during the take home engagement. We had missed (C) above, which we gathered by collaborating with Colorado. We performed additional analyses to validate this finding.

First, we obtained the Taint graph from the "session" parameter passed to the Scalr.log callsite in IBASys.init, whose content is eventually written to file. We selected the control flow touched by the taint graph and obtained an inter-procedural projected control graph (PCG) with these as input events (shown below). This should reveal whether or not there are guard conditions (based on the username) governing the addition of image data to the current session. […]

This PCG reveals 3 branch conditions, all of which are in the client side of the application. This indicates that the server does no validation with respect to the username. Specifically, this confirms the finding that even if a user sends multiple packets containing image data to the

server with differing user names, all the image data is saved to the same current session corresponding to the username in the first packet. […]"

Live-Collaborative Evaluation:

Iowa identified the intended vulnerability through collaborations with Colorado and Northeastern.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.7 Medpedia

A.5.4.2.7.1 Question 009 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"When a page is requested, ContentController.getMasterPage() handles the request, using ArticleService.writeMasterPage() to retrieve the page content from the zim database. By extracting the relevant code, we performed dynamic analysis to enumerate the pages which appear in the "list of all articles" and record the raw content sizes for analysis. Without using the length of the initial page request, there are too many collisions in possible page response lengths to determine the page visited (secret). […] Responses are padded using one of 8 lengths, established when the server starts. The ContentController constructor registers and creates an instance of the custom SecureRandom "com.bbn.crypto.SecureRandom". When a page is later requested, ContentController.getMasterPage() uses the SecureRandom instance to select a number between 32 and 2048, which is the length of the random bytes added in the "X-Padding" attribute. […] The X-Padding lengths can be obtained by an attacker by requesting a page and observing the length of the X-Padding attribute. Because the padding returned is randomly selected, there is a question about how many observations are required before there is a 95% chance of having observed all 8, as required by the operational budget. This is the coupon collector problem (https://en.wikipedia.org/wiki/Coupon_collector's_problem), which when normally posed asks when there is a 50% chance of having made enough observations (for 8 values, approximately 22 observations). To determine the number of observations required for 95% success, a small program was written to simulate 10,000 trials, and 95% of the time, 38-39 active operations is enough to ensure all 8 X-Padding lengths have been observed. Responses are only observable at 16-byte granularity due to AES encryption. Assuming the worst case, the 8 possible pads will cause the 8 largest 16-byte block collisions to further collide with one another. This results in fewer than 250 possible pages in the worst case, which can be resolved with oracle operations."

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Vanderbilt/UCSB contributed observations from Profit. Nico raised other possible sources of noise, including the TLS handshake and chunking. We later decided these could be overcome with engineering; I have noted the change to my cost model as a result. Nico ran additional trials on the SecureRandom instance to demonstrate that it does return values evenly distributed, but it

happens to initialize in such a way to almost always includes 2006 (prior to the server adding 32).

Two Six labs contributed a dynamic simulation that lends confidence that the worst-case scenario is difficult to reproduce, and therefore the attack should be within budget in practice. My revised analysis follows.

The number of oracle queries cited in the original report was based on a program which models the observable sizes of articles accounting for AES 16-byte blocks and random pads, but a flawed manual calculation in post processing resulted in an optimistic estimate. After Nico brought up the “chunking” phenomenon, I adjusted the model to incorporate additional length for articles >= 8192 bytes. It does not account for the \r\n and chunk-sizes but based on other observations the worst-case collisions are occurring between 2000-10000 bytes, and the addition of chunking metadata should usually reduce collisions.

Methodology

The cost model takes as input the association between GET URL lengths and page response lengths. Based on empirical observation, the GET URL length can be inferred from the number of AES blocks. Block boundaries were determined empirically and incorporated into the model (“offset 15” in our cost model code). This allows the analysis to consider only the subset of possible responses based on the length, within 16-byte AES block granularity. Even then, there are over 33,000 articles for URLs in the length $16 \leq x < 31$ range. Next, we factor in the 8 possible random pads, assuming the worst-case scenario wherein the most articles collide. To do this, the model calculates all possible 16-byte windows, and records the number of articles which would fall within the window. The worst case is when the largest 8 windows all map to the same observable size. Selecting disjoint windows maximizes the number of unique articles, the top 4 within 2048 bytes of each other being 41, 41, 35, 32. […]”

Live-Collaborative Evaluation:

Collaboration with Vanderbilt helped to identify sources of noise in observable output that can mask the observable size of the secret. Collaboration with Two Six Labs helped Iowa perform the worst-case analysis. This collaboration helped Iowa to confirm their initial analysis performed during the take-home engagement.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.8 Poker

A.5.4.2.8.1 Question 004 (SC Space, Intended Not Vulnerable, Take-home: Yes, Live: No)

Take-Home Response:

“[…]The method PokerServer.sendTableToClient() is interesting because it reveals a functionality that a human cannot visually observe on reference platform. As the app description says, it has two modes of operation - CLI (command line) and GUI (graphical). In the GUI mode, the app has to display several aspects related to the cards and tables such as the skins of the cards, which involves rendering images. In order to do that, server sends the information of the current state of the table to client. This is a side channel, because although it does not contain information about the opponent’s hole cards, it does contain critical information about the cards the opponent currently has (which is eventually rendered on the GUI). Even in the CLI mode, this information is communicated; it is just that the client just chooses not to display it. Thus, the

network traffic between the server and the victim is an important observable relate to the secret. We proceeded to find the code segments where information on cards are communicated. This network activity happens whenever hole cards are dealt. We observed the loop which sends the information on player's cards to the player (victim's cards to the victim). This loop in PokerServer.sendTableToClient() creates a string corresponding to the card and which is eventually sent over the network (not in this loop, this loop only returns the string created). There is a variable first which is supposed to turn on noise by appending blank spaces but it is never set (if it is false then this noise is added). Hence, there is no noise. To state formally, the length of the string (part of observable request) is the observable, and it is a function of the card name and is thus related to the secret"

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. We don't believe the potential vulnerability reported by is vulnerable. The PokerServer.sendTableToClient() string concatenation adds commas between cards in the outputlist. The sendMessageToClients() method add significant padding noise. This is believed to prevent the attack outlined by Iowa.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In the collaboration, Control Team actually tried to build the exploit based on the outline we and the rest of the teams provided. While building the full exploit, they found out that there is noise due to the random padding added to the messages corresponding to the cards dealt. If the noise was reliably predictable, which we thought it should be since the PRNG in the application does not use different seeds, the side channels we found are strong enough. Unfortunately, that is not the case.

In the collaboration, GrammaTech brought up the possibility of breaking the PRNG to recover the length of the contents of the message by removing the random padding. Control Team discovered that in order to do so on the NUC, you need to run it for a very long time and need to load a large amount of data on the NUC. As per the Operation Definition, the limit on the resource consumption for the exploit on the NUC is one day at a stretch or the memory available. Control team's experiments revealed that it is not possible to achieve the desired outcome of removal of padding within the limits specified in the Operation Definition.

This discovery renders the side channels and the attack outline we proposed is not strong enough and over budget. Thus, we are changing our response from Yes to No."

Live-Collaborative Evaluation:

The collaborative engagement appears to have helped Iowa to better understand the random padding applied to the playing cards. Additionally, Iowa's response reaches a similar conclusion to Vanderbilt, GrammaTech, and Colorado: that the instantiation of new pseudorandom seeds for different threads, coupled with the restrictions of the questions to a single game prevent the attacker from being able to predict the sequence of "random" padding.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.9 Searchable Blog

A.5.4.2.9.1 Question 018 (AC Space, Intended Not Vulnerable, Take-home: No)

Take-Home Response:

"To answer that there is no AC space vulnerability, we consider hypotheses related to increase in memory use due to repeated growth of collections or Strings in the app, or due to recursive calls (that may consume stack space). We explore these hypotheses systematically. Hypothesis 1: Collection add callsites or array assignments grow memory to cause AC Space vulnerability. Refutation: We audit fields of collections and callsites of these, and the array types and assignments to arrays. We also audit loops reachable from the fields and loops containing and branches governing callsites. The one in RankingService puts the file names and their corresponding indices (integer) to a new HashMap each time in the function. Since the number of files is no more than 730 in the app (any new files uploaded are saved in the same name of userblog.html), this collection has a max size of 730. Therefore, this cannot cause an AC Space according to this question. […] There are 13 branches governing the 15 callsites, and 8 loops containing them. We audited all these branches and loops within TitleService and RankingService using the catalog views (branch, loop catalog views shown below). The 8 loops were all monotonic, had only one termination condition and matched simple monotonicity patterns that iterate over an array or collection or APIs that return the file names. We could manually audit the code for the loops and branches and conclude that AC Space per this question cannot be caused by any of them. Of the 13 branches, 6 were termination conditions. Of the remaining 7, one of the branches restricted title keywords to be > 3 characters. Besides that, none of the branches were suspicious because they seemed legitimate conditions for the app. […]"

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.4.2.9.2 Question 027 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The app uses Apache's RealMatrix to calculate and store a weighted graph of the links in the blogs. There are a total of 730 blogs in the app, and whenever a new blog is submitted, the app saves it to the hardcoded file named "userblog.html" (this is included in the 730 blogs). If the file exists, it is overwritten. This contents of this file is the only input the attacker can control through this entry point. […] A survey of the loop catalog revealed a total of 17 loops in the app. One particular loop stood out, because its upper bound suggested that termination of the loop depended on some mathematical property being satisfied. Our CFR Decompiler view recovered the following easyto- understand source code. […] The termination of this loop depends on two conditions, which seem to indicate a convergence of a column of the passed matrix M with respect to the passed vector v upto a configurable precision. A taint analysis revealed that this matrix M corresponds to the field RankingService.A (or A), and the vector v corresponds to RankingService.b (or B). […]"

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“During the take home engagement, we had correctly identified the loop causing the AC Time vulnerability in MatrixRoutines.estimateStationeryVector. We identified two potential vulnerabilities caused by the loop - (1) the convergence could be delayed by setting up the matrix in a specific configuration, and (2) the floating-point operations in each iteration check could be expensive, as per the canonical example.

For (1), we had specifically found that when the matrix used in this loop is a lower triangular matrix, the loop runs long enough to exceed the resource usage limit. However, we were unable to produce the userblog.html file whose upload would trigger such a matrix configuration. This is because the upload would only affect exactly one row or one column of the matrix (the last, corresponding to userblog.html), and a lower triangular matrix requires the modification of other blogs as well. Our original reason to think that the matrix could be set up this way is that the other cells of the matrix do get updated in the iterations of the vulnerable loop.

During our collaboration, Colorado demonstrated an exploit found by their fuzzer. This exploit uses a html file in which useblog.html links itself and has no links to any of the other 729 blogs. Seeing the exploit from Colorado helped us understand that the lower triangular matrix configuration, although sufficient, is not necessary, to trigger the vulnerability.

To summarize, we had enough analysis during the take home engagement to give us confidence that there is a vulnerability, but we were unable to build an exploit within the given operations. The collaboration helped us refine our understanding of how to exploit the vulnerable loop within the allowed operations.”

Live-Collaborative Evaluation:

Iowa confirmed the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.10 SimpleVote 1

A.5.4.2.10.1 Question 019 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“An attacker with voter credentials can login, select an election they are eligible for, and enter a series of three invalid registration keys which will cause the server to use time in excess of the budget (300 seconds), using less than 10kB of input. This blocks the logins of benign users. Exploit: A sequence of registration keys which triggers the vulnerability is:

995: 932193500906

1002: 032103201906

1008: 032103801906”

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.11 SimpleVote 2

A.5.4.2.11.1 Question 005 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Simplevote_2 supports multiple servers which communicate with each other in a peer-to-peer fashion. They exchange vote totals between themselves which includes a summary of the current finalized results in HTML. A user with Voter credentials and a registration key for an election can log into the system, and update their ballot with a specially crafted text response. The text controls an integer threshold, which determines which candidate/choices are mentioned by name in the summary (included if >= a particular percentage of votes). Further, the user controlled text is used by the compression algorithm such that any occurrences of the text are fully compressed; therefore it is advantageous to use the target candidate's name. This results in an observable difference in the peer-to-peer traffic (596 bytes per occurrence), being smaller whenever the candidate's name still appears in the summary. Updating the ballot ten times triggers peer-to-peer communication. We believe within given operations, attacker can find out the percentage of votes, which then can be used to find the best precinct. […]"

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.11.2 Question 016 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"The two branch nodes captured by the PCG above, govern all but 1 event, the callsite to fetchMultipartContents. This method is an argument parser, which is not very interesting from point of view of finding the relation between observable and secret. Following is the code as shown by CFR decompiler view […] The interesting part is that if we enter an incorrect password with a username that exists we would still get back a 'session string'. This is observable from the browser's URL, and it is generated only when we try to log in with an incorrect password but with a username that exists. The question clearly states that attacker knows the username of the victim. Thus, we concluded this must be the observable that may reveal the secret. Every user, whether an admin or not, has a profile. This profile is a HashMap with a set of predefined attributes. The server loads this information when the session begins.

The list is as follows: {name, password, ADMIN, age, city, district, party, username, voterid}. ADMIN attribute (or trait as the app calls these attributes) is only present if the user is administrator. Similarly, age, city, district, party are optional attributes. The only mandatory attribute, apart from name, username and password is voterid. Hence, first we must figure out what other traits are present apart from these three. This can only be observed when user logs in and server loads his/her traits. (This explains why this question requires the victim to log in once before the attack begins.) The second point is the so called profile - which are the optional traits i.e. Age, City etc. have a fixed set of values. This means we only need to figure out the length/size of this HashMap to identify which attributes are present and then we can start using oracle queries to guess the profile. In order to do that we need to strip off username, id and password and name. The name is padded to be of constant length, id is of constant length and username we know. So the secret really is the length of the password. This token comes from processPassword which in turn calls zip before returning it. The loop that generates the token is completely wierd and does unnecessary computation to check the password. For every character checked, it inserts a character from the password at a different offset. In other words, it returns a permutation of the password plus some padding. The zipping further adds some noise to it. The padding depends on the length of the password. The constraints on the length are at least 5 characters and at most 15 characters. Now we have most of the pieces to formulate what we believe is the side channel. […]"

Take-Home Evaluation:

Iowa flagged the vulnerable method used to verify a submitted password and that the return from this method may leak the secret. Iowa defined the secret as the user's profile, not recognizing that the actual secret is the information to view the profile, in this case, the user's password. Iowa's incorrect secret definition led down a path of determining the password length via a weaker side channel, then trying to determine the individual components of the profile page via oracle queries. Iowa should have identified the leak of the secret (password) and then recognized that this is sufficient to log in as the other user and directly view their profile.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"During the collaboration, the Vanderbilt team brought up the possibility of a CRIME style attack. From the information we obtained through the collaboration, all of us reached the same suspicious code. The observation they had was that the randomization does not add noise (in form of padding) as we initially thought, since CRIME attack depends upon the noise being length preserving. In this application, the randomizer always starts with the same seed and thus the length of the noisy padding is always fixed, allowing for a CRIME style attack. Specifically, this means one can build an exploit to guess the password one character at a time.

Based on our prior analysis we know where is the code that is touched when the password is incorrect, in particular LoginManager.grabSessionString and LoginManager.pullTokenByte methods. From the collaboration, we now also understand that for a given length of password string guessed, we get a string of proportional length back. As the padding is predictable in terms of length, we know how many characters we get back. Examining the two methods, we can see it adds the string of modulo 10 value for every character guessed that is incorrect. As modulo 10 value ranges from 0 to 9, it equates to adding a character to the token returned. […]

It means that whenever guess_1 returned a token of length L_1 and guess_2 returned a token of length L_2, and L_1 < L_2, then guess_1 had (L_2 - L_1) number of characters that were additionally correct than the ones in guess_2. So, if we modify our guesses one character at a time till we observe L_2 - L_1 = 1, then we know that the last character modification pointed out to the correct character at that position in the password. Based on this we can build an attack."

Live-Collaborative Evaluation:

Collaboration with Vanderbilt helped Iowa identify the intended vulnerability of a CRIME-style attack.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.12 Stegosaurus

A.5.4.2.12.1 Question 003 (SC Time, Unintended Not Vulnerable, Take-home: Yes, Live: No)

Take-Home Response:

"Stegger.hide() hides the message in the image, processing the message one bit at a time. The server can be queried for progress by an attacker, which is reported in terms of the current message bit being processed. There is a difference between the time taken for processing a one bit and a zero bit, which is a function of the key and image size. For certain keys and image sizes, the side channel is strong enough to observe the values of the bits in terms of the time delays for progress queries. The secret is observable as an attacker can query for progress from the server using a predictable URL that does not use a session id (they do not have to be the user or have their credentials). […]"

Take-Home Evaluation:

Iowa's response contains correct and incorrect statements. The team stated that the server's response time to the heartbeat request is limited to every 50 ms and missed that a race condition can for the intended vulnerability. However, this side channel was determined to be insufficiently strong.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We are changing our answer from "Yes" to "No" following the collaborative engagement. My understanding of the details above is the same, but per the operational definition of a vulnerability, this should be characterized as a "No" because the attacker has less than 95% chance of successfully observing the secret with respect to the space of possible key and image sizes, which are controlled by the benign user (as noted in the first report).

I understood at the time of the first filing of this report that the space of benign user inputs which were vulnerable was a relatively small portion of the input space but did not characterize how small. The following additional details are to support the argument that there is less than 95% probability of exploiting the side channel.

First, note again that the input guards are broken, and image sizes smaller than 64x64 are allowed. In do_post(), the bound check is always false:

```
if (bgh < 64 && bgh > 2048 || bgw < 64 && bgw > 2048) {
```

```
    error = "Please use images larger than 64x64 and smaller than
    2048x2048";
}
```

Unless there is a second guard for images larger than 2048x2048, the user input space for images is not bounded, so the non-vulnerable range would be unbounded as only very small images are vulnerable. For the sake of argument, assume that image sizes are bounded at 2048x2048. […]”

Live-Collaborative Evaluation:

Similar to Two Six Labs, Iowa changed their response after collaboration to the correct answer.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.13 StuffTracker

A.5.4.2.13.1 Question 030 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“StuffTracker has four entry points: item lists can be sent via a POST request to /stuff, which returns a document ID, documents can be requested by document ID via a GET request to /getsdl, the dictionary of possible items can be requested via a GET to /dict, and all posted documents can be searched for items matching a search term via a GET to /stuff. The first and last of these are involved in the exploit. Upon inspecting the decompiled source of com.ainfosec.StuffTracker.parse.ItemListHandler, it is obvious that it fails to validate the XML that it receives, which opens it up to attack: it does things in response to visiting the tags <item>, </item>, </list>, and in response to visiting characters, but never validates that there is only one <list> tag, that <item> tags occur inside a list, and not within each other, or that these are the only XML tags that occur in the input. Upon encountering a <item> tag, an ItemListHandler$State object is created, populated with its long-form ID (in the `fullname` field). A map is checked for a previous <item> tag with the same ID, if one is found, a `pstate` field in the State is pointed to the previous state with the same ID. When characters are encountered, it sets the current state’s owner string to those characters (crashing if the current state is null), then (after unproductively checking whether the current state is null) adds the current State object to an ArrayList stored in the field `accum`. When the corresponding </item> tag is encountered, the current State’s occurrence count is incremented, and the State is added to a map based on its fullname. If <item> tags are never nested, this means that every single State object will have an occurrence count of exactly one, because a new State is allocated when visiting the corresponding <item> tag. By nesting <item> tags, we could cause some of them to have an occurrence count of zero but give others high occurrence counts; if these counts are used, this may give an additional line of attack, but this is not necessary for developing an exploit that meets the budget, given the many other lines of attack available. When a </list> tag is encountered, the method createSDLDoc_and_LocalDictionary() is called, which creates three strings...the first is created using an SDLDoc, and is an XML document corresponding to the items found in `accum`, with their owners. The second is a “local dictionary”, which is intended to contain one copy of every unique ID used in this particular document; it inserts an ID into the local dictionary for each element in `accum` whose `pState` field is null. The third is a list of occurrence counts, which I believe is intended to be the count of each unique ID, but, for a benign user (and likewise for our attacks), is instead a string of spacedelimited ‘1’s, one for each item in `accum`, which may be many more entries than the list in the dictionary, if there were

items in the list with non-null `pState` fields (that is, if there was more than one state with the same ID, and a </item> tag was encountered while the first was the state stored in ItemListHandler's `state` field). These three strings are used to store the document and index it for search purposes, with the resulting document ID being stored in a field. The way these three strings are built up gives us ample ground for attack. By placing the first item with a given ID into `accum` multiple times (by splitting the owner string with ignored XML tags), we are able to not only duplicate that item, but also its ID's entry in the local dictionary […]"

Take-Home Evaluation:

Iowa correctly responded that the challenge program is vulnerable

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.14 Tawa FS

A.5.4.2.14.1 Question 012 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"[…] To delay the application for 50 seconds at least 20 slow file uploads would need to be queued up before the benign request. While the size of the uploaded file would count against the attacker's budget, a very small file could be uploaded very slowly (a 5 byte file could be uploaded at a rate of 1 byte per second with curl's "--limit-rate 1" option). In fact a file upload is not even required, any HTTP request can be sent slowly with this option. With a budget of 40kB, an attacker would have a sufficient budget to perform this attack. Since the attack budget only considers the sum of the PDU bytes sent to the server, there is no limit on the number of bytes received from the server. A request that slowly sends and receives the response of an HTTP request can be formed with just 16 bytes of the attacker's budget by making a request to the index.html page. This is the smallest request that can be sent. NanoHTTPd and the application do not enforce that Host, User-Agent, Accept, or Connection headers be included with the request. A get request without a path is assumed to be "/". A request to "/" is assumed to be "index.html" as evaluated by RequestHandler.process. The response of the index.html is exactly 969 bytes long, which if received at a rate of 1 byte per second by the client corresponds to approximately 16 minutes of processing time. With a budget of 40kB = 40 * 1024 = 40960 bytes, we can execute the request to index.html 40960 / 16 = 2560 times. […]"

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is vulnerable

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.4.2.14.2 Question 020 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"It is possible to create a race condition in the filesystem's defragmentation routine that causes an infinite loop that grows a file by 4kB in each iteration of the loop until the budget is exceeded. […]"

Take-Home Evaluation:

Iowa identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5 Northeastern University

A.5.5.1 Northeastern Overview

Northeastern answered all 31 questions in the engagement with a 55% accuracy for the take-home and an 90% accuracy for the live engagement. Northeastern answered the most questions in the engagement but had the 3rd lowest accuracy in the take-home. Following the live engagement however, the team had the 4th highest accuracy. Northeastern's 35 percentage-point increase in accuracy from the take-home to the live engagement was the 4th highest, and in cases where the teams changed their answer from the take-home engagement to the live engagement they had a 100% accuracy.

In the take-home engagement when Northeastern answered incorrectly, for side channels 57% of the time this was due to a failure to find the side channel; for algorithmic complexity vulnerabilities, 100% of the incorrect responses were due to a failure to identify the complexity. The team had the most difficulty with AC Time questions with a 33% accuracy in the take-home engagement. During the take-home engagement we (the EL) identified three examples, Accounting Wizard Questions 014 and 023 and SimpleVote Question 019, where the team was looking for direct attacker control and known vulnerabilities from experience/previous engagement; as a result, the team missed the attacker's ability to indirectly control the complexity. In Poker Question 31, Northeastern's take-home response showed that the team sampled the cards of different suits. It does not appear that this sampling spanned the class of secret symbols (rendering a card vs loading a card from cache). We hypothesized that Northeastern would benefit greatly from the collaborative engagement as 85% of their incorrect responses were due to the team answering "No" after missing the intended vulnerability.

Following the collaborative engagement, the team achieved a 100% accuracy in 3 of the 5 question categories.

A.5.5.1.1 Take-Home Engagement Scores

Table A-97: Engagement 5 NEU Take Home Engagement Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	5	56	5	56
SC Time	6	4	67	4	67
SC Space/ Time	1	1	100	1	100
AC in Space	9	5	56	5	56
AC in Time	6	2	33	2	33
Total	**31**	**17**	**55**	**17**	**55**

Table A-98: Engagement 5 NEU Take Home Engagement Vulnerabuluty Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	24	12	**50**	12	**50**
Not Vulnerable	7	5	**71**	5	**71**
Yes Answer	14	12	**86**	12	**86**
No Answer	17	5	**29**	5	**29**

A.5.5.1.2 Live Engagement Scores

Table A-99: Engagement 5 NEU Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	7	78
SC Time	6	5	83
SC Space/ Time	1	1	100
AC in Space	9	9	100
AC in Time	6	6	100
Total	**31**	**28**	**90**

Table A-100: Engagement 5 NEU Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	24	22	**92**
Not Vulnerable	7	6	**86**
Yes Answer	23	22	**96**
No Answer	8	6	**75**

A.5.5.2 Northeastern Responses

A.5.5.2.1 Accounting Wizard

A.5.5.2.1.1 Question 011 (SC Time, Unintended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"There are at least two ways to figure at the project's expenditure limit for a project with just employee credentials. The simpler way is kind of a side-channel but not a timing one.
When you add hours to a project, there is a check if they would fit in the expenditure limit and are only added in that case. You can then check the currently accounted hours, and so we can just do a binary search here. The expenditure range is from 1.000 to 3.000.000 and we got 30.000 operations. Try to set the hours to 1.500.000 (h = 1.500.000) and check if it worked (cost 1)

a. Yes: Try h = 2.250.00
b. No: Try h = 7.500.00

Continue this binary search until we know the exact hours with a precision of 100, which is the cost of 1 hour. Start asking the oracle in binary search in between the 100-dollar range. Finally, we will hit the exact expenditure.
The second way is based on a real timing side-channel, which is due to the loop that traverses the list of items in the Project to get the latest id in newHours() of FileStore.java:

```
for (BaseObject bo : project.getChildren()) {
        long id;
        if (!(bo instanceof Hours) || (id = bo.getId().longValue()) <= next) continue;
        next = id;
}
```

That means if we add hours to the project, the list will be traversed and if it is long enough we'll see a timing difference.
So here is the attack:
1. Add a lot of Items to the project, e.g. 3000 items. This depends on the precision of your time Measurements
2. Then do the same binary search as in the first attack. However, instead of checking the current hours, you simply measure the timing of the request and check if the loop was traversed or not

a. If the hours didn't fit the following check will fail and the loop won't be traversed if (FileStore.hoursFit(project, userHours, employeeHours))
b. If the hours fit, the loop will be traversed and you'll see the timing difference

3. Continue the binary search with the oracle as in the first attack

The binary search is completely feasible with 30.000 operations and the critical part of the second attack is only how many operations have to be spent on filling the list for an accurate measurement. Note that both attacks also rely on the fact that the hours can be reset to 0."

Take-Home Evaluation:

There is an unintended vulnerability discovered by Northeastern and several other teams in the live-collaborative engagement. The strength of this unintended vulnerability can be amplified using the unintended AC Space vulnerability. While Northeastern's take-home response does not indicate the amplification, the remainder of the operational budget for this question can be used to achieve the same amplification effect.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"It is possible to obtain the project's expenditure limit by setting the hours and checking the currently accounted hours. If the accounted hours did not change, it means that the limit has been reached. However, we consider this attack out of scope, since it is not a timing side-channel. Our side-channel attack is based on the fact that the command updateHours takes a different amount of time if the added hours exceed the expenditure limit. This difference can be amplified if the log functionality of the program is set to trace (and a lot of items are present). To set the log to trace it is necessary to generate an exception while setting a new Locale, which can be done by trying to set the Locale to a not-existing value. The following is our (unoptimized) POV. Our code needs about 300 requests to setup the program, maximum 6000 queries to have an approximation of the limit, and maximum 300 requests to query the oracle and get a precise value (within 10$) of the limit."

Live-Collaborative Evaluation:

This builds on the Take-Home evaluation. There was a mistake in packaging this challenge program: the application came provided with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects' contents or states.

The unintended side-channel vulnerability identified by Northeastern and several STAC teams was confirmed.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"The employee has no control over the size of the files in the .store folder except for the item name, which cannot be used for creating big files given the limits in the budgets. The employee has some control over the content of the log file, but all the employee-controllable strings written in the log file need to be sent every time by the employee. This makes it impossible to exceed the file size limit within the budget."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attacker can cause a custom error to be thrown triggering more verbose logging.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"It is possible to increase the size of the file .accounting.log to make it bigger than 1MB. To do so, it is first necessary to set the log in trace mode. This can be achieved by generating an exception while setting the Locale. In turn, this exception can be generated by setting the Locale to a non-existing value. Then, an attacker can just add multiple objects (100) to the project, increasing (quadratically) the log size. To remain within budget, we merged connections to the program within a single SSL connection."

Live-Collaborative Evaluation:

Northeastern identified the intended vulnerability

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.1.3 Question 023 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"The code that builds the json from the current Project is efficient and has no complexity vulnerability. In addition, we believe that there is no json injection-style attack."

Take-Home Evaluation:

The challenge contains an intended vulnerability. The attacker can submit items to cause the budget-settling algorithm to exceed the resource usage limit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"There is an algorithm complexity vulnerability in the generation of the Expenditure Report. In particular, if items with specific properties are present the method getAllowedExpenditures will run for more than the allowed time (500s). […]"

Live-Collaborative Evaluation:

Northeastern identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We found an attack based on the size of a project file on the disk. An attacker can do the following:

1. Login (cost: 1)
2. Retrieve the sizes of the files in '/report/files' using the specific command (cost: 1)
3. Add an item (content doesn't matter) (cost: 1)
4. Retrieve again the sizes of the files in '/report/files' (cost: 1)
5. Check for a file whose size increased by 44 bytes
    a. That is your Project-File (say "PF")

b. Do the following calculation to retrieve the number of budget items B:
    i. B = (size(PF) - length(project_name) - 24)//44
    ii. Submit B to the oracle (cost 1)

This attack has a total cost of 5, which is within the budget."

Take-Home Evaluation:

Correctly identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.2 BattleBoats 1

A.5.5.2.2.1 Question 006 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Our loop analysis spotted quite a few unbounded loops (or loops that are actually bounded but the analysis fails to figure out the bounds). We analyzed them and did not find any loop that can does not terminate. After we worked on other questions in battleboats_2, we came back and revisited the do-while loop inside the function com.cyberpointllc.stac.battleship.StrikeLocator.calculateStrikeTime(), and we realized that although this loop is bounded by 25 (which is a constant ITERATIONS_TO_ALLOW), it is still possible to forge a request that takes a very long time to finish a single iteration. The BigDecimal class in Java faithfully simulates arithmetic operations on decimals of arbitrary precision, and since all the arithmetic operations probably have a complexity of O(N) where N is the number of digits of the decimal, we should be able to feed it a *very precise* decimal to exploit this algorithmic complexity vulnerability and get a very long execution time for each iteration.

The exploit is basically `shoot 0 1e-2000000000 0 0` after starting a game and finishing placing boats on the board. It easily exceeds the runtime limit of 120 seconds."

Take-Home Evaluation:

This challenge contains an intended AC Time vulnerability in calculateStrikeTime(). This vulnerability allows the attacker to enter input values such that the height never reaches zero via negative input height. Northeastern's exploit appears to trigger the vulnerability but in an indirect way, causing the same end result that the code is not able to reach the termination condition for calculateStrikeTime().

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.2.2 Question 028 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"We carefully annotated the cost in terms of the sizes of network packets generated from the shoot command and performed a cost analysis. The calculated cost does not seem to correlate to whether the shot misses or hits the board. We also monitored the network traffic of real games, and packet sizes do not change whether an attack misses or hits the board. Therefore, it does not seem to be possible to deduce the location from observing the sizes of network traffic flows. We believe it is impossible for a player to discover the exact coordinates of their own cannon on an opponent's ocean board."

Take-Home Evaluation:

Northeastern correctly determined that the challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.3 BattleBoats 2

A.5.5.2.3.1 Question 010 (SC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The source of the timing side channel is in com.cyberpointllc.stac.combatboat.Competition.isOver(), which introduces a hard 30 ms of delay whenever this function is called. Since in com.cyberpointllc.stac.comatboat.OceanPanel , this function is only called when the calculated pixel is not on the ocean panel, we are able to use this small time difference to infer whether a cannon shot hits the ocean panel or not. This time difference can be observed by monitoring network traffic. Thus, we believe it is possible for the attacker to discover the exact coordinates of their own cannon on the ocean board of the opponent."

Take-Home Evaluation:

Northeastern correctly identified the intended vulnerability

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.3.2 Question 021 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Our loop analysis found many potentially unbounded loops in this program. While manually analyzing the unbounded loops and understanding the program's behavior, we realized that the do-while loop in com.cyberpointllc.stac.combatboat.HitLocator.computeHitTime() is indeed unbounded. We reversed the entire class, and we realized that it is essentially solving a quadratic equation that describes a trajectory using the Newton-Raphson method. The exit condition of that do-while loop is $|tPrevious - tCurrent| <= OPTIMUM\_DELTA$ or $|currentHeight| <= EPSILON$. It is known to all (or most people) that the Newton-Raphson method takes a long time to converge when the derivative is small, or it enters an infinite recursion (overshoot?). The latter is handled by the $if\ (iterations > ITERATIONS\_TO\_ALLOW)$ check, which essentially stores all trajectory data into a hashmap variable. Therefore, a possible attack vector is to make the derivative very small in the quadratic equation that we are trying to solve, which in turn makes the loop iterate many times and put as many values into the hashmap variable as possible, and in the end fills up the memory. We verified it with command `shoot 15 98.$(python -c "print '0'*9000")1 90 0` (replace the shell-python part with an actual string of zeros), and we successfully forced the other client to exceed the resource usage limit of 800 MB of memory."

Take-Home Evaluation:

Northeastern identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.4 BraidIt 1

A.5.5.2.4.1 Question 002 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"There seems to be no timing side-channel to gain an advantage in the game. We found two possible timing relevant functions, between the victim sending the modified Plait and the end of the round. First of all the setStep() function has an unnecessary triple-while loop […] However, this seems to be a red herring as the timing of the function is not leaking any information nor does the delay help the attacker in any way. The second interesting find is the freeSimplify() function […] The recursive call gives a certain time penalty when checking for the equality of two plaits. However, it is called AFTER the attacker has already called make_guess. So, this also does not give an advantage to the attacker."

Take-Home Evaluation:

The challenge did not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.4.2 Question 007 (SC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"The attack is very similar to the one in Question_001, which uses the difference in Packet sizes when receiving the modified Plait, due to the format string in getPlaitString() (called by grabWeaveString in Question_001) […]

The attack works in the following way:

1. Initiate the Game
2. Select sizes (values don't really matter) e.g. plait_sizes = [1, 1, 1, 1, 1] (cost 1)
3. Receive the selection (mod_plait) and monitor the traffic (cost 1)
4. Check the last packet's payload size and do the following calculation:
    a. Base_size = 123
    b. plait_sum = sum(ps for ps in plait_sizes)
    c. Mod_index = (payload_length - base_size - plait_sum) / 50
5. Mod_index is the selected plait's index
    a. There is one exception, if the the payload_length - plait_sum = base_size. This could also mean that Plait is unmodified.
6. Submit index to oracle (cost 1)
    a. If exception occurred and the answer was wrong resend the index of the plait that is similar to the modified one to the oracle (cost 1)
7. With this technique the attacker wins with 100% probability and the cost is at most 4."

Take-Home Evaluation:

The intended vulnerability requires the attacker to trigger the side channel by sending an out-of-range braid length. Northeastern's exploit instead uses the string format algorithm for turning braids into strings to send to "win the game". It is unclear whether this side channel is sufficient to answer the question of "determine the selected and modified braid" (different from "win the game"). Additionally, the cost calculations appear to count cost as command into the attacker console. Cost is per command received by the benign user's peer application. The vulnerability discovered by Northeastern differs slightly from the intended one but fits within the operational budget.

Post-Take-Home Analysis Ruling: Correct

A.5.5.2.4.3 Question 017 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"The only file written to the filesystem is the log file. The only way we found that could "spam" the log file would be an unbounded recursion in the method freeSimplify(). This only happens

when the guess is compared against the choice, however, which only happens on the attacker (not the victim) side."

Take-Home Evaluation:

The challenge contains an intended vulnerability. The challenge has a while loop in calculating braid reduction. The conditional check to exit the while loop can be made to not terminate for certain conditions, causing the while loop to not terminate. This loop logs to disk with each iteration.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Braidit_1 contains an algorithmic complexity vulnerability in space in its handling of certain braid configurations during its scoring procedure. A PoC was constructed that exhibits this vulnerability.

Braidit_1 logs messages to disk during processing of game messages. An attacker can modify a braid using a malicious client that causes the victim to execute an excessive number of iterations of the braid simplification algorithm. Each iteration of the algorithm logs at least one message to disk, causing the resource limit to be exceeded within the input budget.

The PoC is constructed as follows. First, an attacker accepts a game offer from a victim. Selecting one of the braids, the attacker uses the modified client to return a long braid value that was found via fuzzing. This is returned to the victim, whose client then executes the simplification function. That is, Plait.isEquivalent() is invoked, which calls Plain.isEmpty(), which then invokes Plait.simplifyCompletely(). In this method, simplification is repeatedly performed while Plait.isSimplified() does not evaluate to true. In this while loop, Plait.simplifyOnce() is called and logs the current state of the plait to disk. Additionally, Plait.freeSimplify() is also invoked within the loop and also logs the plait state to disk. The combination of these two method invocations in this while loop causes the resource limit to be exceeded."

Live-Collaborative Evaluation:

Northeastern identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.5 BraidIt 2

A.5.5.2.5.1 Question 001 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We found a space side-channel in the ModifiedBraidMessage() method. The length message consists of each of the braids as a simple string plus the selected message in a specific format, which is built in the WeaveElectedState.grabWeaveString() function:

```
public String grabWeaveString() {
        String weaveStr = this.weave.pullIntersections();
        return String.format("%1$-" + this.weaveCount * weaveStr.length() + "s", weaveStr);
}
```

Where weaveCount is the index (starting from 1) So the attacker can do the following in Round 1 and 3:

1. Initiate the Game
2. Select sizes (values don't really matter) e.g. braid_size = [1, 1, 1, 1, 1] (cost 1)
3. Receive the selection (mod_braid) and monitor the traffic (cost 1)
4. Check the last packet's payload size and do the following calculation:
    a. Base_size = 68
    b. Braid_sum = sum(bs for bs in braid_sizes)
    c. Mod_length = length(mod_braid)
    d. Mod_index = (payload_length - base_size - braid_sum) / mod_length
5. Mod_index is the selected braid's index
6. Submit index to oracle (cost 1)

With this technique the attacker wins with 100% probability and the cost is at most 3."

Take-Home Evaluation:

Northeastern identified a side channel sufficiently similar to the intended vulnerability. This is given a post engagement analysis ruling of correct, but as with BraidIt 1, the team appears to count operations as input to the attacker's console application not inputs to the benign user's remote application. Additionally, the secret is to determine the selected and modified braid not to win the game. These are different objectives.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.5.2 Question 022 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"There seems to be no algorithmic complexity vulnerability. The Strands, Weave and Braid lengths are all bounded in size and the operations defined on a weave are all efficient. The best chance for an attacker to exploit the program would be to create a 'bad' braid-configuration such that the target would suffer a complexity vulnerability when modifying the selected braid. However, this doesn't seem to be possible due to the bounds described above."

Take-Home Evaluation:

The challenge contains an intended vulnerability in the braid reduction algorithm. There is a while loop for which the method used to check the termination condition always returns false for certain inputs. This causes the while loop to not terminate.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Braidit_2 contains an algorithmic complexity vulnerability in time due to excessive iterations of the braid simplification algorithm upon reception of a maliciously modified braid. Specifically, the execution time limit can reliably be exceeded under the input budget. ‘The PoC is identical to the one for braidit_1 question 17; please refer to the justification for more details.”

Live-Collaborative Evaluation:

The intended vulnerability differs slightly from the one in BraidIt 1 Question 017, but there do exist overlapping exploits.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.6 IBASys

A.5.5.2.6.1 Question 015 (SC Space, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“After careful analysis, we do not think there is a side channel in space that allows an attacker to impersonate another user. An attacker can try to login as another user, but an incorrect guess always results in the generation of a random 128-bit key independent of the secret of that user.”

Take-Home Evaluation:

Northeastern missed the intended vulnerability that allows the attacker to piggyback off a well-formatted request with a mal-formatted request. The return packet for the mal-formatted request allows the attacker to leak the secret bitstring bit-by-bit and by extension passcode image.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.5.2.6.2 Question 024 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“A user's 128-bit key is encoded into the least significant bits of pixels contained in the first 100x100 area of their password image. During authentication, the system crops the image to this area, extracts the key, and compares it with the user’s key stored in the database. As soon as the application detects a difference between the key given in the image and the key stored in the database, it returns as unsuccessful. Probing the application with our own images revealed that an adversary only needs to learn 32 bits of the 128-bit key stored in the database to successfully authenticate as a user. In isolation, this authentication loop that compares user input directly with secret data is a textbook example of a side channel in time that an adversary could probe to recover bits of a user’s stored key.

Obtaining a signal for this side-channel is complicated by time spent decrypting, parsing, cropping the image, and communicating the results between threads. Probing the application with an image that fails authentication immediately and with one that has a complete password fails to reveal a measurable timing difference from the attacker’s perspective over the network. If

an attacker cannot distinguish a completely unsuccessful attempt from a successful one using the arrival times of the application's responses, then this timing information will not help it guess individual bits of the secret key."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.6.3 Question 029 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Our fuzzer helped in answering this question. Common requests typically do not trigger writes to disk, but fuzzing the application revealed packets that cause XML files to appear in the logs folder. On its own, this is not enough to demonstrate a vulnerability, but the tool can separately show that submitting multiple requests under the same AES key will cause a single session to grow, even if requests are made under different usernames. This gives us enough information to construct an exploit that incrementally grows the session under a single key and forces the cumulative session to be written to disk for each request. In the end, the adversary will store enough files such that the cumulative size exceeds the resource usage limit of 150 MB."

Take-Home Evaluation:

Northeastern identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.7 Medpedia

A.5.5.2.7.1 Question 009 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The exact size of the response can be computed by breaking the custom PRNG. The size of the initial listing is known to the attacker, disclosing the current random number. Once the seed has been reversed from the current number the next one can be computed. The attack can be carried out this way:

1. Retrieve the list and record links and the whole request/response size
2. Fetch all articles and record their size
3. Intercept a legitimate request/response and infer the current random number
4. Intercept the following requests/responses predicting the random number
5. Build a subset of possible matching articles and submit them to the oracle

If step 2 cannot be carried out the attack can still be feasible. The fetch phase is spread over the first N interceptions, and after those the attack can proceed as described from step 3."

Take-Home Evaluation:

Northeastern identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"A side channel in space exists in this application that leaks both the article URL (and thus its name) as well as its length. Despite several sources of noise, we have constructed a PoC that can reliably infer articles accessed by victims within the input budget.

The medpedia application uses a custom PRNG that is installed by a Spring application controller. The output of this PRNG is extremely weak, taking on only 8 values. One PRNG instance is shared between all user sessions, and thus while the exact set of eight values are not predictable a priori, once the application has been started any user can sample the application to recover the entire PRNG range for that execution.

The PRNG is used to pad HTTP responses by adding an X-Padding header of length PRNG() % 2016 + 32. If the PRNG were strong, this would render space side-channels difficult to exploit by a passive adversary. However, since there are only eight possible paddings and these eight values are known to any application user, space side channels are not severely degraded by this noise source.

An additional source of noise is possible depending on the specific TLS session cipher suite negotiated by the victim and application. If a stream cipher is negotiated, then encrypted data payloads have the same size as the plaintext input and therefore no noise is added. However, if a block cipher is selected such as AES then noise in the form of padding up to the cipher blocksize will be present. For block ciphers used in current TLS implementations, this is bounded by 16.

A final source of noise is present in the HTTP responses returned by medpedia that depends on the size of the article. That is, above a certain size, articles will be returned using HTTP chunked transfer encoding. However, the additional chunking metadata overhead (several newlines and a hexadecimal length per chunk and the difference in HTTP headers to account for the different transfer encoding) is essentially a constant factor that can be precomputed based on known article sizes.

The PoC operates in the following fashion. First, the attacker precomputes a map of article URL lengths to a vector of articles. The attacker then observes and records the victim's interactions with the application, namely obtaining a listing of all articles and then in a subsequent request directly accessing one of those articles. The attacker identifies the packets corresponding to the article HTTPS request and responses; despite the fact that these are encrypted, they are trivial to identify based on packet size and sequence.

Using the observed (cipher blocksize-padded) article URL length, 16 article bins are selected. The attacker then collects all matching articles and selects all those which a length that matches the observed (padded) article length with any possible combination of cipher blocksize, PRNG response header, and transfer encoding padding. The resulting list is then simply submitted to the oracle.

In our testing, using HTTPS TCP keepalive allowed us to reliably recover all eight PRNG values using one TCP connection. Despite the fact that HTTPS sessions were torn down after approximately 50 requests, this session length implies that not observing a PRNG value occurs with vanishingly low probability. Article collisions due to padding-adjusted article size matchings across the 16 bins could exceed the operational budget in theory; however, in our empirical testing (long-run repeated executions of the PoC over the article set) success was obtained well above the required threshold."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.8 Poker

A.5.5.2.8.1 Question 004 (SC Space, Intended Not Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"This challenge is very difficult to solve, as it requires an intensive observation effort and some sophisticated calculation. We noticed that there is a clear size difference in the size of the cards and this difference is clearly a side channel in space. However, we were not able to bring the proof of concept to the required level of accuracy. We achieved at best 50% accuracy when 95% is the goal.

With more time it might be possible to improve the attack to achieve the desired accuracy, and we therefore believe the vulnerability is present in this challenge program."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability, and we currently do not believe than an unintended vulnerability satisfying the operation budget has been discovered.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change

Live-Collaborative Evaluation:

No change

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.5.2.8.2 Question 031 (SC Time, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“There are only minor timing differences in the rendering of the cards for the table image, but these differences do not translate well into a signal in network delays. Therefore, it is not possible to predict the cards above a 95% probability.”

Take-Home Evaluation:

There is a time side channel that allows an attacker to differentiate dealt vs un-dealt cards following a change of the card skin. An attacker can segment the deck in half, playing enough games to ensure that half the cards have been dealt under the new skin and half have not. The attacker can then use the time side channel in a new game with the benign user to determine which segment the benign user’s card falls in. Oracle queries can be used to resolve the remaining entropy of the secret.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.5.2.9 Searchable Blog

A.5.5.2.9.1 Question 018 (AC Space, Intended Not Vulnerable, Take-home: No)

Take-Home Response:

“There are no data structures in memory that would expand beyond the resource usage limit given the input budget. The only data structures in the program are simple matrices and hash maps with lists to map keywords to files, and both do not appear to support the necessary expansion of ~500x (4M -> 2G).”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.5.2.9.2 Question 027 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Assumption: The attacker request immediately precedes the benign request. The benign request is sent within roughly one second of the attacker’s request.

By carefully crafting an HTML file that contains a link to “/blog/userblog.html”, the attacker can force the server to spend an excessive amount of time in the “performRanking” method and therefore the response to the user will exceed the time budget (10 seconds).”

Take-Home Evaluation:

Northeastern identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.10 SimpleVote 1

A.5.5.2.10.1 Question 019 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“We ran a cost analysis on the handlePost() function in the com.cyberpointllc.stac.webmaster.handler.LoginGuide class, and we believe that the timing cost of the entire call tree is bounded. There is no infinite loop or other construct that will cause a timeout. Hence we do not believe there is any vulnerability present.”

Take-Home Evaluation:

An attacker can cause an increase in the allowed maximum size of an input key. A longer input key can be used to exceed the resource usage limit for a benign request sent directly after the attacker request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Initially we firmly believed that the attacker must trigger the algorithmic complexity vulnerability through the specified benign request, which is the login interface in this question. But the fact that Simplevote_1 is a single-threaded webserver opens an interesting attack vector to this question: Attacker can trigger an algorithmic complexity vulnerability somewhere else as a logged-in user, and then other benign users cannot perform login request within the resource usage limit (300 s). With this in mind, it is very easy to figure out the attack: The recursive calculation performed inside the FunkyFunction class on registration key provided by users can be very expensive. Obviously, the name FunkyFunction led us to scrutinize that class, but the fact that there are unbounded recursive method calls (our loop analysis fails to bound the recursion call) and BigDecimal s are used without any limit on their precision also hint that this “funky” class might cause troubles. Two integers are extracted from the registration key, and one of them is used to perform the recursive calculation defined in PermissionAuthenticator.confirmEssence() . We then manually analyzed the code, realizing that although the calculation is accelerated by the predefined values in the cache file, a user may increase the upper bound of the input number, PermissionAuthenticator.MAX_N , by 7 each time the input number equals to MAX_N . Naturally, the calculation results are not cached for integers greater than the initial MAX_N , which is 995. The attack is basically increasing the upper bound to a bigger number (in our tests, we increased it to 995 + 7 * 3 = 1016), and then trigger the calculation by a smaller number (like 1015).”

Live-Collaborative Evaluation:

Northeastern identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.10.2 Question 025 (SC Time/Space, Unintended Not Vuln., Take-home: No, Live: No)

Take-Home Response:

"It is unclear what the role of the attacker is (e.g, is the attacker also a voter, or is he the other server?), but we also simply did not find any possible space-related side-channel attacks that would reveal the password of a voter or reveal the full ballot of that voter. However, we are less confident on this answer, since there might be attack vectors that we did not identify."

Take-Home Evaluation:

The challenge program contained an intended vulnerability whereby a registration key is transformed to an integer in a deterministic manner, and the runtime for processing a registration key is linearly correlated to the value of the key. The side channel strength is dependent on the secret range, and the application code must unambiguously provide this. Our aim was to have a 1-to-1 mapping between a registration key and a translated integer value of the key. However, in the application code 3 specific characters in the registration key are replaced with 3 specific numbers during the transformation from registration key input to an integer. The challenge program therefore allows for 2^6 possible registration keys to map to the same integer. There are not enough oracle queries to resolve this. This unintended not-vulnerable case was identified by Vanderbilt.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.11 SimpleVote 2

A.5.5.2.11.1 Question 005 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Through manual analysis (our automatic cost analysis ran into problems with this challenge program), we realized that ReportsMessageCreator.buildMessage() delivers carefully padded strings for everything, regardless of the number of finalized votes for any election. In addition, the compression in LZW is broken, which essentially creates a "summary" of the original election summary output in HTML with every character expanded to four characters. We cannot see any side-channel in space in the sizes of network packets delivered from one server to the other when they are synchronizing election results. Therefore, we believe the described vulnerability does not exist in this challenge program."

Take-Home Evaluation:

Northeastern analyzed the synchronization packet traffic for a side channel leak and found none. This is somewhat correct as the intended SC vulnerability must be triggered. An attacker can craft inputs to influence the compression scheme in the synchronization messages. This can be used to perform a binary search over the secret space.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“The formation variable that is fed to LZW.zip() is controllable by user input. If the value of formation equals to the name of one of the candidates, the compression algorithm will then remove that name from the compression output, which essentially reduces the size of the output (note on the network the output is encrypted, so the size of the output is the only side channel we can rely on). The attack is basically providing a candidate name as the input to formation , then observe the size of the communication traffic between the servers. Since normally the entire answer "tally" is properly padded and its size can be used as a baseline, when a collision between formation and a candidate's name occurs, there will be a significant reduction in the size of the communication. We can then use this information to figure out the information that this question requires.

When we were working alone, we noticed that formation was user-controllable, and it was used inside the LZW.zip() method. We did not realize that the formation could impact the output of zip(), which was pretty silly of us.”

Live-Collaborative Evaluation:

Northeastern identified the intended vulnerability

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.11.2 Question 008 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“Through our analysis, it seems that the only place that a space complexity vulnerability could ever be triggered is through the decompression functions in LZW. But, these decompression functions are never called anywhere in this challenge program.”

Take-Home Evaluation:

Northeastern’s response indicates that time was spent analyzing the decompression algorithms. Previous engagements have included vulnerabilities in decompression. The intended vulnerability however is in the recursive question/answer structure used to maintain election responses. An attacker can setup a cyclical structure causing continual recursion when trying to store data.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“The LZW.zip() method may potentially consume a huge amount of memory for two reasons: a) A non-optimal formation (code word mapping) is used for compression (it essentially expands every character into four characters), and b) strings created by calling substring() are not GC’ed in time. This can only be seen from the control flow graph visualization of LZW.zip() method in angr Management, because we cannot successfully decompile that method. To trigger this algorithmic complexity vulnerability in space, we need to build a string as long as possible and with the same characters as many times as possible, and then feed the string into zip. By analyzing the control flow graph, we know that the zip() method is only used when server A gathers the votes and sends the results to server B. Because of the crazy padding that simplevote does in the ReportsMessageCreator.buildMessage() method before transmitting the votes, we can

easily create a desirable long string by submitting invalid votes multiple times (so the long string is mostly filled with spaces) and then trigger the vulnerability inside zip() .”

Live-Collaborative Evaluation:

Northeastern did not flag the intended vulnerability but has flagged an unintended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.11.3 Question 016 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“The only way to obtain the profile of another voter is to infer his password. This does not seem to be feasible by exploiting a space side channel from the login page.”

Take-Home Evaluation:

The challenge program contains a vulnerability in compression that allows for a CRIME-style attack to leak a user’s password.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Obviously, the attack vector is to leak the password of the target user. In method TokenedPasswordChecker.processPassword() , each character from the password that user inputs is inserted into the token variable, which is then immediately followed by the character at the corresponding location of the correct password. Although there is padding, the random variable controlling the padding behavior is seeded by the CRC32 result of the correct password, which means it is not really "random", and is in fact always the same for a user as long as he does not modify his password. Since then the token that contains interleaved user-input password and correct password is compressed using Deflater.deflate() in method TokenedPasswordChecker.zip() , and the length of this token variable is preserved when it is passed back to the user (via `id=` in the URL after a failed authentication, while individual bytes inside the token variable are totally overriden), this opens the possibility to commit a CRIME attack. We can simply guess at each location of the password, what the correct password character is, and if we got a prefix correct, the returned "id=" will have one less character than otherwise, due to how zip deflating works. We are then able to use this approach to recover the entire password of that user byte by byte.”

Live-Collaborative Evaluation:

Northeastern identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.12 STAC SQL

A.5.5.2.12.1 Question 013 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“The logfile appears to be safe: it always contains roughly the same data as the input command, and the toString() implementations of the actions seem reasonable. The only suspicious code we found allows one to severely increase the size of the database disproportionally to the user input size by creating a table of strings, inserting many rows with very small values to keep the query

size short. Then using an update you can update all rows at once. However, in our testing there was no way to approach a 2MB database size because the input size of the update string is limited to ~970 characters. Trying to balance this out by making multiple columns also did not work well because you have to repeat the long string in the update statements for each column, which severely limits the available options. In conclusion, neither from optimizing nor from random and manual testing were we able to ever exceed 1MB in size. The best result we obtained, by using 216 rows, 8 cols and 480 characters of string, was 1019691 bytes."

Take-Home Evaluation:

The challenge contains an intended vulnerability in the compression scheme via Huffman coding tree.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Through collaboration with other teams (Colorado, Vanderbilt, and GrammaTech), we realized that there are two possible attack vectors. The first one is using the attack we proposed originally (inserting small values and then updating everything with one command), but using the Huffman encoding as well as sending everything all together (instead of sending command one by one) to reduce the overhead in traffic, hence reducing the PDU. We are then able to make the database file reach 2MB in size. The second attack vector is to increase the size of the log file, which we originally believed to be safe. The cause of this attack is a bug inside SQLInsertAction.buildFromTokens() : It does not break out of the switch/case when it matches a left parenthesis, which causes the parser to incorrectly enter the "values" keyword branch. This allows us to set columns and values individually, with two different insert SQL commands inserting to a non-existent table (so it prints out error messages in the log file). Then by running "insert help" for over 500 times, we can dump many error messages containing the long column names and value strings we set earlier into the log file, which makes the size of the log file go beyond the limit."

Live-Collaborative Evaluation:

The challenge contains an intended vulnerability and after collaborating with Vanderbilt and GrammaTech Northeastern arrived at the correct result.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.13 Stegosaurus

A.5.5.2.13.1 Question 003 (SC Time, Unintended Not Vulnerable, Take-home: Yes, Live: No)

Take-Home Response:

"We found a timing side-channel in the Stegger.hide() function. […] Note that this code loops based on the value of pkCopy and maxOffset, where maxOffset is derived from the image dimensions and pkCopy is calculated from the secret key, the current character in the message, and the previous offset. The loop is effectively doing the same as the following modulo operation and, therefore, the first modulo operation is unnecessary. However, it does introduce a heavy timing penalty due to the large exponentiation of the current message bit. In addition, the loop is completely missing from the corresponding part in the complementary find code. If using the

time penalty introduced by this loop together with the information from the "/progress.html" web interface when hiding an image, we can leak the secret message.

The following describes an attack that would leak the message based on the available operations from the question and the input or output image:

1. Basic information
    a. Value is the current message bit in the current message byte, that allows for only 8 values:
        i. 0->3, 2->6, 4->12, 8->24, 16->48, 32->96, 64->192, 128->384
    b. Exponentiated with 80, those values are multiple magnitudes different from each other
    c. The value of maxOffset is static for the particular image
    d. The value of this.perf is completely static
    e. That means the runtime of the loop is directly proportional to the value of the "value" variable
    f. Note that the value of key is negligible small to make a difference in the initial value of pkCopy
    g. Also note that the initial offset is always 0
2. Analysis (Total Cost: 127)
    a. We obtain the timing for each ASCII byte value by doing the following:
        i. Create a message with enough bytes that we receive at least two heartbeats (h0, h1) of the particular byte
            1. An option would be to modify the client-side javascript to pull for the status more often and get a finer-granularity measurement
        ii. Calculate the time for the byte as $t\_byte\_x = (h1.time - h0.time) / ((h1.val - h0.val) / 8)$
    b. Create a map for each ASCII value:
        i. $Map[ascii\_char] = t\_byte\_x$
3. Attack
    a. Given the maximum size of a message, we can pre-compute the unique timing for each message and store these in a map (cost: 0)
    b. Then we obtain the target's timing information (cost: 1)
    c. We then simply retrieve the message from the map (cost: 0)"

Take-Home Evaluation:

Northeastern identified that there is a timing difference in hiding bit 1 vs bit 0. It is unclear if their proposed exploit satisfies the operational budget. Additionally, while they noted the ability to get information from the "/progress.html" interface it is not clear that they recognized the race condition allow bit-by-bit leakage of the secret. However, the team flagged the individual components of the side channel and responded that the program was vulnerable. This intended SC vulnerability was discovered to be of insufficient strength.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We found a timing side-channel in the Stegger.hide() function. Specifically, in lines 116-120 […] Note that this code loops based on the value of pkCopy and maxOffset, where maxOffset is derived from the image dimensions and pkCopy is calculated from the secret key, the current

character in the message, and the previous offset. The loop is effectively doing the same as the following modulo operation and, therefore, the first modulo operation is unnecessary. However, it does introduce a timing penalty due to the large exponentiation of the current message bit. In addition, the loop is completely missing from the corresponding part in the complementary find code. By using the time penalty introduced by this loop together with the information from the "/progress.html" web interface when hiding an image, we could leak the secret message. However, after collaborating with other teams, we believe that this side-channel is NOT exploitable within budget. This is mainly due to the fact that we cannot do assumptions on the properties of the image."

Live-Collaborative Evaluation:

As with the take-home engagement, Northeastern flagged the SC Time vulnerability in the Stegger.hide() function. Northeastern noted that after collaboration, the exploit proposed in the take-home was found to exceed the operational budget.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.14 StuffTracker

A.5.5.2.14.1 Question 030 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We observed that stufftracker uses SAX to parse XML and utilized a known space complexity attack against SAX (with a specially crafted XML payload) to make it use more than the maximum allowed amount of memory."

Take-Home Evaluation:

The challenge contains an intended vulnerability in the SDL formatting scheme. Northeastern identified an unintended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.15 Tawa FS

A.5.5.2.15.1 Question 012 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"All operations between the attacker and the benign user are fairly well separated through the user session. The user and the attacker work on different filesystems. Therefore, an attacker cannot fragment or otherwise impact a user's filesystem. Technically, it is possible to fill the partition on NUC's SSD that stores the filesystems (/tmp): 40KB is the input budget and the minimum request to write 100MB is 65 bytes, thus allowing ~615 requests, which is ~60GB.

However, the NUC's root partition is 50GB. However, even this did not increase runtime of user requests significantly. Instead, user requests seemed to fail entirely (empty response via curl)."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. A recursive function in the defragmentation algorithm can be made the self-recurse and delay a benign user's request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"There is a vulnerability in the directory traversal implementation (com.bbn.Filesystem.findAllFiles) that can be used to create an infinite loop if a cycle in directories is created due to the way that paths are being parsed. Creating a cycle is possible since when creating directories, there is no verification if the directory name contains the separation character /, while findAllFiles separates pathnames based on it. Since tawa_fs is limited to two threads, invoking defragment twice after setup of the vulnerability results in both threads being locked up. […]

There is also an alternative vulnerability, though likely unintended: rate-limiting the attacker's connection to 1b/s, opening ~50 connections (that fetch the index page) to fill up the thread pool and request queue. If a benign user now requests any page, then the response by the server will take longer than the budget allows. Since it is possible to minimize the attacker's request size, it falls within the budget."

Live-Collaborative Evaluation:

Working with Colorado, Northeastern identified the intended vulnerability. Northeastern also flagged an unintended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.5.2.15.2 Question 020 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"There are two race conditions that seem reasonable to fit into the budget and result in a filesystem exceeded the resource usage limit:

Sketch of the POC:

1. Create a fragmented filesystem (upload small x2, delete first small, upload big)
2. Trigger a defragmentation operation (via command) [Thread 1]
3. Trigger race condition #1 by uploading another file that uses previously free inodes that were attempted to be used for defragmentation [Thread 2]:
   This forces [Thread 1] to set cleanMove to false, which throws a RuntimeException, which is propagated up the stack. The exception triggers backupInode() to run automatically in [Thread 1].
4. During the execution of backupInode() in [Thread 1], trigger race condition #2 by overwriting the file that is being defragmented [Thread 2]: After `sourceBytes = srcBlk.getBytes()` has been set, overwrite the inodes of the inode that is being moved during defragmentation of [Thread 1] (big) (overwrite filename or delete and upload new file). This results in a divergence between sourceBytes and the content at filesInodes[blockId] , which are compared in slowEquals(), and which results in

errorByte being set to < 4096 (filesInodes is initialized early and not touched later). It then executes the loop infinitely many times, because sourceBytes is not read again and the file on disk differs. Part of the loop is writing the current block to the backupRegion , however backupRegion is incremented for each loop counter, i.e., each iteration of the loop increases the disk space by 1 block at the end of the filesystem and extends it through `RandomAccessFile.write(byte)` by 4096 bytes. That is, after roughly ~400,000 iterations of the loop, 1.5GB logical disk space should be used."

Take-Home Evaluation:

Northeastern identified the intended vulnerability

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Collaboration with Iowa gave insight into an easier way to trigger the vulnerability, which is by repeated calls to defragment."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6 University of Maryland

A.5.6.1 UMD Overview

UMD answered 22 questions in the engagement with a 36% accuracy in the take-home and an 86% accuracy in the live engagement. The team's 50 percentage-point increase in accuracy from the take-home to the live engagement was the 2nd highest increase in accuracy. During the take-home engagement, the team had the most difficulty with SC and AC Space questions. The UMD's take-home "No" answers had the least accuracy (15%) of all the categories for the team. UMD had a 100% accuracy on questions where they changed their answer from the take-home to the live engagement.

A.5.6.1.1 Take-Home Engagement Scores

Table A-101: Engagement 5 UMD Take Home Engagement Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	6	2	33	2	33
SC Time	4	2	50	1	25
SC Space/ Time	0	0	0	0	0
AC in Space	7	3	43	2	29
AC in Time	5	3	60	3	60
Total	**22**	**10**	**45**	**8**	**36**

Table A-102: Engagement 5 UMD Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	19	8	**39**	6	**33**
Not Vulnerable	3	2	**50**	2	**50**
Yes Answer	9	8	**78**	6	**67**
No Answer	13	2	**15**	2	**15**

A.5.6.1.2 Live Engagement Scores

Table A-103: Engagement 5 UMD Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	6	4	67
SC Time	4	4	100
SC Space/ Time	0	0	0
AC in Space	7	6	86
AC in Time	5	5	100
Total	**22**	**19**	**86**

Table A-104: Engagement 5 UMD Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	19	16	**84**
Not Vulnerable	3	3	**100**
Yes Answer	17	16	**94**
No Answer	5	3	**60**

A.5.6.2 UMD Responses

A.5.6.2.1 Accounting Wizard

A.5.6.2.1.1 Question 011 (SC Time, Unintended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We decompiled the program and ran static taint analysis on it. We manually identified the inputs for public and secret values. The static taint analysis failed, perhaps because of the use of lambda functions for the server handler. Instead, by hand we traced the control flow from the employee entry point as well as the flow of the secret information.

Even though we do not have an explicit exploit for this problem, we highly suspect two places where an attacker can gain information about the allowed expenditure. The first way is related to

the method getAffordableLines() (see Question 23). It's possible that a list of items can be added to the project, of increasing value, and use the way vectorDec considers subsets of the set of those items, to find when the sum of the items in the subset is less than the allowed expenditure. We believe that 30,000 operations are enough to repeat this procedure multiple times.

The second way is more subtle and related to the method updateHours(), where additional steps are performed if the cost of the updated hours does not exceed the allowed expenditure, and the time it takes to perform these steps is detectable by the attacker."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. However, Iowa and Northeastern identified an unintended vulnerability during the take-home where under certain conditions, an attacker-controlled input of hours is directly compared against the secret. This initially weak side channel can be amplified using either the intended AC Space vulnerability or by using the remaining input budget.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In our previous justification, we described two possible locations an exploit could take advantage of. It seems the second one, using updateHours to find the project expenditure limit is the more promising way. We did not provide an explicit exploit last time, because it was not entirely clear to us whether the timing difference observed (not on the reference platform) was observable enough to succeed 85% of the time as the question requires. Other teams had reached a similar conclusion. During the discussions, we attempted amplifying the difference, and the first way we tried is by taking advantage of the vectorDec function. We did not succeed this way, and the other method used was to combine it with another vulnerable piece of code. The idea (mostly by Vanderbilt we believe) was to use the locale exploit discovered during a different accounting_wizard attack (by GrammaTech), and tried changing the locale to an unsupported string, which caused the logging to be set at the TRACE level (after catching an exception). This way we feel a binary search attack is possible using the updateHours method, by trying different times, and in between such calls, update the hours to 0 (to delete them). When newHours is called, and when hours do fit in the project expenditure, the method BaseObject.write is called, which in turn calls BaseObjectMarshaller.toBytes, which goes through all chidlren and performs some logging operations (only when the level is at TRACE). This way, by having enough items in the project, the logging takes enough time, to cause an amplification of difference in time. This amplification is enough to make the timing difference observable, and it seems the success probability of this measurement is a lot higher than 85%. With even a few binary search steps, one can get to a level after which they can try oracle queries for each individual value. […]"

Live-Collaborative Evaluation:

There was a mistake in packaging this challenge program: the application came provided with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects' contents or states.

The unintended vulnerability discovered by Northeastern and Iowa and amplified during collaboration was confirmed.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We decompiled the program. Whenever hours are updated, an object is stored on file with the username (not sent in the request). If the username were large, this could provide enough amplification of the request."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attacker can cause a custom except to be thrown. This results in verbose logging.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"One of the teams (Grammatech) for this problem figured out that if one tries setting the locale to an unsupported one, then an exception is thrown that sets the logging to TRACE level. This way by adding multiple items into a project (assuming there is one), logging happens during the execution of BaseObject.write() because of the for loop in BaseObjectMarshaller.toBytes(). (please see justification of question 11 for additional details)."

Live-Collaborative Evaluation:

UMD identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.1.3 Question 023 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We decompiled the program and we run dynamic analysis on a simplified version of the method getAffordableLines(). This method goes through all the subsets of the set of items ordered until it finds one whose sum of costs does not exceed the allowed expenditure. By adding lots of items whose cost does not exceed the allowed expenditure, but any two of whose combined cost does exceed the allowed expenditure, makes the loop run for a long time."

Take-Home Evaluation:

UMD identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"We do not change our answer on this question. Some other teams seem to agree with this exploit."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We decompiled the program and ran static taint analysis on it. We manually identified the inputs for public and secret values. The static taint analysis failed, perhaps because of the use of lambda functions for the server handler. Instead, by hand we traced the control flow from the employee entry point as well as the flow of the secret information. We identified where the outputs are and found that getHours and report/files can return more data than just a status code. getHours is the only command with a flow that involves the project. The flow from this project information to the response output only involves the employee's own hours."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The sizes of the files written to disk (retrievable via a user request) can be used to leak the secret.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After collaboration with other teams it seemed pretty clear that there is a vulnerability. In particular there is a clear correlation between the file sizes and the secret. One of the teams figured out that the correlation is ((project-file-size) -24 - (project-name-length))/44."

Live-Collaborative Evaluation:

UMD appears to have identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.2 BattleBoats 1

A.5.6.2.2.1 Question 006 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We decompiled the program and ran the fuzzer with a harness aiming for the method calculateStrikeTime(). It appears negative values for the player's move are allowed, and the program runs indefinitely. Move send 'shoot -10 -10 -10 -10'"

Take-Home Evaluation:

UMD identified the intended vulnerability

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"We do not change our answer on this question, which was found with our fuzzer and confirmed manually. After collaboration with others, we found agreement for our exploit."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.3 BattleBoats 2

A.5.6.2.3.1 Question 021 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We decompiled the program and run dynamic analysis on the method computeHitTime(), to find what initial values would make the Newton method iteration procedure run longer. Setting the initial vertical speed to be 98, causes enough iterations through the loop, and since the precision of the BigDecimals is high, there are rarely any collisions in the datastructure. As a result, if this move is sent enough times by a player, the allowed memory is exceeded."

Take-Home Evaluation:

UMD flagged the vulnerable method, however it does not appear from the description that they identified the intended vulnerability. The exploit provided was tested with attempts to send repeatedly; however, the resource usage limit was not exceeded.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In our previous justification, we described two possible locations an exploit could take advantage of. It seems the second one, using updateHours to find the project expenditure limit is the more promising way. We did not provide an explicit exploit last time, because it was not entirely clear to us whether the timing difference observed (not on the reference platform) was observable enough to succeed 85% of the time as the question requires. Other teams had reached a similar conclusion. During the discussions, we attempted amplifying the difference, and the first way we tried is by taking advantage of the vectorDec function. We did not succeed this way, and the other method used was to combine it with another vulnerable piece of code. The idea (mostly by Vanderbilt we believe) was to use the locale exploit discovered during a different accountin_wizard attack (by Grammatech), and tried changing the locale to an unsupported string, which caused the logging to be set at the TRACE level (after catching an exception). This way we feel a binary search attack is possible using the updateHours method, by trying different times, and in between such calls, update the hours to 0 (to delete them). When newHours is called, and when hours do fit in the project expenditure, the method BaseObject.write is called, which in turn calls BaseObjectMarshaller.toBytes, which goes through all chidlren and performes some logging operations (only when the level is at TRACE). This way, by having enough items in the project, the logging takes enough time, to cause an amplification of difference in time. This amplification is enough to make the timing difference observable, and it seems the success probability of this measurement is a lot higher than 85%. With even a few binary search steps, one can get to a level after which they can try oracle queries for each individual value. […]"

Live-Collaborative Evaluation:

This response appears to be from Accounting Wizard Question 011.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.6.2.4 IBASys

A.5.6.2.4.1 Question 015 (SC Space, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"We decompiled the program and ran static taint analysis on it. All messages are padded to the same length, and the only information that is revealed is a bound on the size of the image. Since the first 100 pixels of the image are all that ultimately matter, the total size is irrelevant. Message comparisons were done using packet captures, extracting the packets sizes, and diffing the results with a correct image versus an incorrect image."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. A race condition allows an attacker to piggyback a malformed request off an invalid but well-formed request to leak the passcode string. This in conjunction with the verification algorithm can be used to construct and input image that will pass verification.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.6.2.4.2 Question 024 (SC Time, Intended Not Vulnerable, Take-home: Yes, Live: No)

Take-Home Response:

"We decompiled the program and ran static taint analysis on it. Through inspecting the code, we found that only the first 100 pixels of the image are relevant. Images are compared based on the parity of the pixel value, not the value itself, so we effectively have a bitmap of length 100. Since the comparison terminates at the first mismatch, it should theoretically be possible to send images with the relevant pixels set to a current parity-based guess and observe the timings. On the platform we tested on, there was enough variability to preclude this timing attack, but that might not be the case on a slower processor and over a higher-latency interface than loopback. Since the minimum acceptable image size will require 35 packets, and we have 3500 operations, discriminating one bit at a time would exhaust our budget without being able to consult the oracle. However, we would expect to hit strings of already-correct bits often (probability 0.5 on each bit to test), so we should need only half the budget, on average. 95% probability of success should only require a comparable fraction of the budget."

Take-Home Evaluation:

The application does not contain an intended timing vulnerability. We don't believe that the vulnerability flagged by UMD is exploitable within the operational budget on the reference platform/network environment which has a higher latency and potentially slower clock speed that the one UMD used to test this potential vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Previously our justification suggested that because the loop to determine the validity of a token would stop after processing the correct number of bitmaps we hypothesized that there was a potential side channel based on the response time. However, there are at most 32 comparisons

and this is not enough spread to change the execution time significantly enough to be noticeable over the network."

Live-Collaborative Evaluation:

The challenge does not contain an intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.4.3 Question 029 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We decompiled the program. Through direct examination of the decompiled code and testing, the only way we could find to generate output files was to send a null image (replayed headers and 10000 null bytes), which causes an error log to be written in the logs/ subdirectory. The largest we were able to make this with the given budget was 17MB. Further, we examined the size of the /proc/<pid>/fd directory (following symlinks), which can yield up to around 90MB on our test platform. This 107MB total is still well-shy of the 150MB target."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The persistence of a session object can be used to submit a malformed request that bypasses the input guard to trigger an ArrayIndexOutOfBounds exception that causes excessive logging to disk.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"When a user uploads data with all zeros an exception is raised and a log file is generated by the session data in memory. However, doing it again would cause the same file. So the exploit works as follows, upload valid data with username A. This generates an AES key. Then modify the client to use the some AES key, a different username B, and data of all zeros. User A's session is still active and the request is authenticated with the AES key. However, the log file generated is named by the hashcode of username B. This interaction is added to the session and the log file is written. Repeat the process with user C and then a new log file will be generated with all the session data up to that point."

Live-Collaborative Evaluation:

UMD identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.5 Medpedia

A.5.6.2.5.1 Question 009 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

""We decompiled the program. When a user requests an article the response HTTP request has an X-Padding Header which adds a random string of length *random.nextInt(2016) + 32*. This means there is no SC because each response will have a random length.""

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The random call does not have sufficient entropy. The attacker can determine the possible paddings using active operations and use the relationship of html page size to article to determine the secret.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Originally, we thought that the random number generator used by the application was the built-in java implementation. However, after collaboration we learned that it was in fact provided by the application. This RNG each time would create 8 numbers with a high probability of 3006 being one of them. The exploit then becomes polling the server with an article of a known length, enough times to learn the size of the padding for each of the 8 random numbers. Then a passive operation of watching the victim request an article. This article could then be one of 8 sizes, found by subtracting the 8 numbers from the size observed. Binning the articles by size you then learn that the article is in one of 8 buckets. You then query the oracle for each article in each bucket. From collaboration we learned that polling of the server to find the 8 pads, is an example of the coupon collection problem. Thus, it takes 39 requests to find all 8 with high probability. Also, the distribution of the size of buckets means the budget for querying is 8 * average number per bucket. This total is with a very high probability within budget to learn the article.”

Live-Collaborative Evaluation:

UMD identified the intended vulnerability through collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.6 Poker

A.5.6.2.6.1 Question 004 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“We decompiled the program and ran static taint analysis on it. The static taint analysis was killed after a 5-hour timeout with no results. Instead, by hand we traced the control flow from attacker input to where the control operated on the victim's secret cards. If the taint analysis worked, it could have revealed how certain commands trigger redrawing the image for both clients simultaneously (via updateClientViews), which in turn redraws the image. A side channel in space could be feasible if there were a relationship between the output image and the pair of secret cards. The padding field appears to be an attempt to mitigate this.

It was difficult to create a harness for the fuzzer, because there is a fairly specific set of interactions between both clients and server necessary to perform to get the server into a state to begin fuzzing on. Another challenge is that the fuzzer would need to coordinate commands between two clients, while our setup is currently designed around a signal client and server. Once setup, the fuzzer can continually measure the output size, with padding, for the same secret cards.”

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“We identified a similar potential exploit in our previous justification and believed that the output padding added enough noise. And after discussion the difficulties of overcoming the padding with other teams, we maintain the same "No" as our previous submission. In particular, the RNG was potentially vulnerable, but not believed to be enough to reduce the search space.”

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.6.2 Question 031 (SC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“We decompiled the program and ran static taint analysis on it. The static taint analysis was killed after a 5-hour timeout with no results. Instead, by hand we traced the control flow from attacker input to where the control operated on the victim's secret cards. If the taint analysis worked, it could have revealed how certain commands trigger redrawing the image for both clients simultaneously (via updateClientViews), which in turn redraws the image. A timing channel would be feasible if the drawing of card had timing specific to each card. The image redraw method (Renderer.drawTable), however, keeps a cache of card images to avoid regenerating them. The setSkin command can be used by the attacker to empty this cache. The reskin card method, however, redraws every card, caching only the desired card. This appears to be a mitigation against a timing channel.

It was difficult to create a harness for the fuzzer, because there is a fairly specific set of interactions between both clients and server necessary to perform to get the server into a state to begin fuzzing on. Another challenge is that the fuzzer would need to coordinate commands between two clients, while our setup is currently designed around a signal client and server. Once setup, the fuzzer could continually try changing skins and issuing other commands to cause the application to redraw the image. Since changing the skins empties the card image cache, this forces the server to redraw the card, and the fuzzer could test whether that side channel signap is strong enough for the parameters of the attack."”

Take-Home Evaluation:

There is a time side channel that allows an attacker to differentiate dealt vs un-dealt cards following a change of the card skin. An attacker can segment the deck in half, playing enough games to ensure that half the cards have been dealt under the new skin and half have not. The attacker can then use the time side channel in a new game with the benign user to determine which segment the benign user’s card falls in. Oracle queries can be used to resolve the remaining entropy of the secret.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“In our previous justification, we originally identified the potential timing channel due to the redrawing of cards after setting the skin. While in collaborations the agreement was generally no, we received confirmation that the redrawing of the cards was the correct source of the timing

leak. Given the ability of the attacker to change skin, we believe this timing channel is strong enough for an exploit."

Live-Collaborative Evaluation:

UMD following collaboration identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.7 Searchable Blog

A.5.6.2.7.1 Question 018 (AC Space, Intended Not Vulnerable, Take-home: No)

Take-Home Response:

"We decompiled the program and run the fuzzer using harnesses and dynamic analysis on specific, and at times simplified parts of the program, but we could not find an algorithmic attack in space."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.6.2.7.2 Question 027 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We decompiled the program and the fuzzer was used in parts to assist with finding the vulnerability. The convergence of the algorithm in the method estimateStationaryVector() can be influenced by a single column in the matrix, and by adding their own blog, the user can determine the last column of the matrix. Uploading a blog that links only to itself, causes the calculation of the rank vector to take a long time."

Take-Home Evaluation:

UMD identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"We did not change our answer from the previous justification. After collaboration, it seems the other teams with a "yes" answer reached the same conclusion as us, given in the previous justification."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.8 SimpleVote 1

A.5.6.2.8.1 Question 019 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“We decompiled the program and ran static taint analysis, looking at functions where login happens. After static analysis of the routines involved in login, we could not identify a vulnerability.”

Take-Home Evaluation:

The challenge program contains an intended AC Time vulnerability. The attacker can cause the bounds of the registration key to be expanded. This can then be used to submit a key that takes a long time to process and can be used to delay a benign user’s request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“We had identified long potential running times in the code related to funkyFunction, but our interpretation of the question had ruled out that code as reachable from the pre-authentication login path. It turns out that interpretation of the question was incorrect, so then if the attacker can reach funkyFunction, the running time is exponential.”

Live-Collaborative Evaluation:

UMD elaborated in this response that a mistaken interpretation of the challenge question led to an incorrect response. However, prior to this engagement, example programs demonstrating the coming changes in the definition and how it would impact changes in the challenge question were released. Additionally, the take-home response did not contain enough information to make this mistake clear.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.9 SimpleVote 2

A.5.6.2.9.1 Question 005 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“After decompilation and static taint analysis we identified a path by which the attacker can send a packet to supply a trie and then that trie is used to compress ballot data, so the size will vary based on the ballot data and the attacker supplied trie.”

Take-Home Evaluation:

UMD correctly determined that the attacker is able to control the data compression. The attacker can control the compression of the synchronization packets sent between the simple vote servers. UMD failed to indicate the relationship between the resulting observable and the secret, and how the attacker is able to manipulate the compression scheme to trigger a side channel. The attacker can trigger the compression scheme to leak whether a target candidate has received at least x% of the ballot in their best precinct. The attacker can then use this scheme to implement a binary search. Additionally, the team does not mention that the attacker can then trigger synchronization between the simple vote servers with 10 ballot submissions to perform each binary search operation.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“We saw another teams exploit, they use the same conditions that we identified but through manual analysis carried the attack through end to end where we did not, which increases our confidence in our initial conclusion.”

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.9.2 Question 016 (SC Space, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“After decompilation and static taint analysis we determined that there is no vulnerability, session ID generation is done securely as far as we can tell.”

Take-Home Evaluation:

The challenge program contains a vulnerability to a CRIME-style attack. This can be used to leak a user’s password.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.6.2.10 STAC SQL

A.5.6.2.10.1 Question 013 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“We decompiled the program and use manual analysis to search for the flow of control for the attacker to submit data. A data flow analysis would have assisted in showing that an update flows to SQLRow.updateFieldData, which calls setData on the field. To reach the file system, this data flows through a compression library (huffman encoding). Since compression is vulnerable to amplification: if the encoding is not recomputed for new data, an attacker can select new bytes to write that are stored with greater than a byte according to the encoding. We plan to use a harness around the compression library's updateTree method to find specific inputs automatically that cause this amplification.”

Take-Home Evaluation:

UMD identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“We did not change the answer, but in collaboration we were more confident in our original answer of yes.”

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.11 Stegosaurus

A.5.6.2.11.1 Question 003 (SC Time, Unintended Not Vulnerable, Take-home: Yes, Live: Yes*)*

Take-Home Response:

"We decompiled the program and ran static taint analysis on it. Upon initial inspection it seemed that the responses had a fixed length. However, we were able to use the fuzzer to mock a chain of interactions and measure the size of network responses and found variation in the responses. This was done by making every method and field public to allow us to set up the session objects the way a real http request would.

Our fuzzer normally uses coverage to score a run, but we added a scorer that used the output of the run. We then fixed the image and the secret message, while varying the secret key used in encryption. Since we found variation in the sizes of response that was dependent on the secret key we found a side channel. However, since our fuzzer was first constructed with complexity attacks in mind we are not yet able to correlate the changes in size with the secret key.

Our manual inspection also found a direct leak. A user submits image to be hidden (encrypted) It remains on server until it someone sends: *GET /image.png?id=<id>*. Until that point an attacker can issue *GET /done.html?id=<id>* to get the secret message. We found that if the attacker spams this request, it will get the secret message while the image is being encrypted."

Take-Home Evaluation:

UMD flagged an unintended out-of-scope vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"No. There is a small side channel by polling the progress of an the encryption of a message. Longer responses are when the algorithm is encrypting the a 1 bit of the message. The number of iterations of the inner loop (the loop for encrypting a single bit) is the secret_key/(3 * height * width * c). Thus, this means that as the picture used increases the time to encrypt a single bit decreases. This speed becomes to fast to poll the server to gain information. When polling the server you must first send the request, then the heartbeat thread is polled for the progress. This thread sleeps for 50 ms and updates the progress. With an increased image size and the speed of which zeros are encrypted any series of zeros would be processed during this sleep. Furthermore, the message is at most 512 Bytes long. Even with a small image, this would mean 4096 requests, which is out of budget."

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.12 StuffTracker

A.5.6.2.12.1 Question 030 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“We decompiled the program. We didn't think the taint analysis would be useful since this is a complexity attack with no secrets and most likely it would mark all methods. We thought the parsing code, the custom-written storage serialization code and some caching code might be suspicious, but did not have time to investigate each of them closely.”

Take-Home Evaluation:

The challenge program contains an intended vulnerability in the SDL parsing scheme. An attacker can upload data that exceeds the resource usage limit during decompression.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“It seems there is a "billion laughs" vulnerability (Colorado). Another exploit given by another team (Draper, Grammatech) was related to providing an XML file with nested lists. There is a check in ItemListHandler.endElement for elements called list, but not in ItemListHandler.startElement and this causes the problem.”

Live-Collaborative Evaluation:

UMD reported the same unintended vulnerability and Colorado and Northeastern.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.13 Tawa FS

A.5.6.2.13.1 Question 012 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“We decompiled the program. We could not find an explicit exploit for this question, but we do think the defragmentation process contains suspicious code. In particular, in the case of a non clean move, (e.g. if there are not enough free continuous blocks), then an attempt is made to backup the inode. The backup procedure, contains a loop that could run as long as the writeBytes procedure throws and exception or fails to write the correct bytes. As an example, if the underlying filesystem is corrupted, this may run indefinitely. Another place that looks suspicious is in CRC.hash(), which seems a O(N^2) algorithm. However, the N here is at most 4096, so it doesn't seem to trigger a real time complexity vulnerability.”

Take-Home Evaluation:

A self-recursive scheme during defragmentation can be triggered by an attacker to delay a benign user’s request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“In our justification, we found one suspicious code in defragment, but couldn't find the exploit. After discussing with collaborators, we confirm this is part of the vulnerability. In fact, the program has several vulnerable implementations. To exploit them, the first one is that the server

employs a fixed pool containing only two threads. Therefore, if both of them are blocked, the server will be blocked. A simple exploit would be to slowly issue GETs to the server, one byte every second. Since the server doesn't have a TIMEOUT for the GETs, this method will result in two threads getting blocked. (Vul 0)

Beyond the above simple strategy, there are indeed vulnerabilities in the implementation. There are three places. First, the mkdir command should call Filesystem.autoCreateFolder, but Filesystem.createFolder is called instead. When a folder with name "x/x" is created using the later method, an inode with the name "x/x" is created. But it will "never" be accessed, except by another vulnerability explained below. (Vul 1)

Second, the Filesystem.findAllFiles function involves a recursive call to itself. This itself is OK if the third vulnerability is not presented, but it will be used to form infinite recursive calls. This function will be called by Filesystem.defragment(), which will be called under the defragment command. (Vul 2)

Third, the FileTable.findInode method has several implementation bugs. The basic flow of findInode is that (1) it finds the inode for the root; (2) it splits a path by '/'; (3) for each folder in the path, it will trace down from the root inode to the particular inode wanted.

When tracing down, if a sub-folder's name is not present in its parent's children list, a NoSuchFileException should be thrown; but the verification logic here is wrong. In L57-58, it checks

```
if (!cMatchBlk.getName().trim().equals(cfolder)) {
  throw new NoSuchFileException(path);       }
```

However, if a sub-folder is not found, cMatchBlk will just remain the same. In the worst-case, e.g., when looking for /x/x, if there is no sub-folder x under /x/, then the cMatchBlk will remain to be the same to the block for /x, and in this case, the above checking will pass through. (Vul 3.1)

If the above statement is not entirely terrible, the rest one is. The method uses a variable inode to keep track of found inode, which will be returned in the end. However, it is only modified during an inner for-loop over all sub-folders (L50), rather than when a match-inode is found (L52-54)!. Therefore, this inode always refers to a wrong inode, when the above vulnerability for a mis-match is triggered. (Vul 3.2) […]"

Live-Collaborative Evaluation:

UMD identified a vulnerability within the application.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.6.2.13.2 Question 020 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We decompiled the program. We could not find an explicit exploit for this question, but we do think the defragmentation process contains suspicious code. In particular, in the case of a non clean move, (e.g. if there are not enough free continuous blocks), then an attempt is made to backup the inode. The backup procedure, contains a loop that could run as long as the writeBytes procedure throws and exception or fails to write the correct bytes. As an example, if the

underlying filesystem is corrupted, this may run indefinitely, and in the process, keep writing bytes on the disk."

Take-Home Evaluation:

The challenge contains an intended vulnerability. A race condition during defragmentation involving the clean move and backup inode calls can be used to cause the termination condition to never be satisfied, consuming all the available disk space.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In our previous justification, we found a suspicious loop that could result in infinite time, but could not discover a way to trigger the conditions that led to that. In collaboration with other teams (Iowa and Grammatech teams had the exact exploit), we discovered that a data race between two threads could trigger that condition, so we change our answer to yes. The problem lies in backupInode, and the while loop in there. The loop will exit only when the slowEqual command finds that the two chunks are the same. Having another thread defraging at the same time, could result in overwritting the same piece of memory (where the chunk is backed up) and cause the slowEquals method to never return 4096 (the exit condition of the loop). This way the program keeps writting data exceeding the allowed limit."

Live-Collaborative Evaluation:

UMD identified the intended vulnerability by confirming what they hinted at in their take-home response.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7 University of Utah

A.5.7.1 Utah Overview

Utah answered 7 questions during the take-home engagement with 86% accuracy. Utah was the only team to identify the intended AC Space vulnerability in Accounting Wizard during the take-home engagement. During the live collaborative engagement, the team's accuracy increased to 92%, with the team answering 5 additional questions.

A.5.7.1.1 Take-Home Engagement Scores

Table A-105: Engagement 5 Utah Take Home Engagement Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	1	100	1	100
SC Time	1	0	0	0	0
SC Space/ Time	0	0	0	0	0
AC in Space	3	3	100	3	100
AC in Time	2	2	100	2	100
Total	**7**	**6**	**86**	**6**	**86**

Table A-106: Engagement 5 Utah Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	5	5	**100**	5	**100**
Not Vulnerable	2	1	**50**	1	**50**
Yes Answer	6	5	**83**	5	**83**
No Answer	1	1	**100**	1	**100**

A.5.7.1.2 Live Engagement Scores

Table A-107: Engagemtn 5 Utah Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	1	1	100
SC Time	3	2	67
SC Space/ Time	0	0	0
AC in Space	5	5	100
AC in Time	3	3	100
Total	**12**	**11**	**92**

Table A-108: Engagement 5 Utah Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	10	9	**90**
Not Vulnerable	2	2	**100**
Yes Answer	10	9	**90**
No Answer	2	2	**100**

A.5.7.2 Utah Responses

A.5.7.2.1 Accounting Wizard

A.5.7.2.1.1 Question 011 (SC Time, Intended Not Vulnerable, Live: Yes)

Live-Collaborative Response:

"The side channel exploits BudgetSolver.getAffordableLines, which uses BudgetSolver.vectorDec to iterate over possible combinations of items to include. This function thus takes time exponential in the number of items in the list. However, getAffordableLines is called only if the total of all items is more than the budget. […] When the binary search range is narrow enough, use oracle queries to exhaustively search in the remaining range."

Live-Collaborative Evaluation:

There was a mistake in packaging this challenge program: the application came provided with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects' contents or states.

In the case of the vulnerability reported by Utah and Colorado, this was investigated prior to the engagement and is not believed to be sufficiently strong. There is a SC Time vulnerability that leaks the residual budget. The Project Expenditure Limit is the secret; the Adjusted Expenditure Limit is the Project Expenditure Limit minus the hours and items charged by other users. By combining the time differential for an employee's total hours being under vs over the Project Expenditure Limit with the intended AC Space vulnerability an attacker can cause a sufficient time differential such that a binary search is feasible. The binary search depends on the attacker's ability to set the total hours charged on a project to zero. As the attacker is an employee, they are only able to set their own total hours charged to zero. As a result, this side channel only leaks the Adjusted Expenditure Limit and not the required Project Expenditure Limit.

Several STAC teams identified an unintended vulnerability during the live collaborative engagement. In the case where an employee has no hours, an attempt to add hours causes an exception to be thrown. In handling the exception, the newly submitted hours are directly compared to the secret Project Expenditure Limit in FileStore.hoursFit() method. This comparison triggers writes to the log file if the newly submitted hours are less than or equal to the secret Project Expenditure Limit. The number of log writes that occurs controls the strength of the side channel. As delivered, the number of log writes is insufficient to distinguish when newly submitted hours exceed versus don't exceed the secret Project Expenditure Limit. However, an intended AC Space vulnerability where verbose logging is triggered can be combined with the addition of more items to a project to increase the number of log writes.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.7.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"The employee can set the locale. When an invalid locale is requested, the logger changes its logging level to TRACE. This causes a lot of information to be logged, but specifically interesting is that every time the employee orders a new item, each previously-ordered item will be printed to the log. This causes the log file to grow quadratically as more items are ordered. After attempting an invalid locale switch, the employee sends 100 unique item ordering requests. We estimated each request to be no more than 500 bytes (as a high upper limit). Through tests, we have found that this will cause the .accounting.log file to increase from practically nothing to over 1 MB with less than 50 kB of transmitted data. The viability of this attack is limited by the length of the name of the project to which the employee has access."

Take-Home Evaluation:

Utah identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.1.3 Question 023 (AC Time, Intended Vulnerable, Live: Yes)

Live-Collaborative Response:

“The attack exploits BudgetSolver.getAffordableLines, which uses BudgetSolver.vectorDec to iterate over possible combinations of items to include. This function thus takes time exponential in the number of items in the list. […]”

Live-Collaborative Evaluation:

Utah identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“The base size of project files is 24 bytes, plus the length of the project's name. Each child of the project, i.e. the hours reports and the items ordered, increases the project's total size by 44 bytes each (40 bytes for the UUID of the child file plus 4 bytes to encode the length of the UUID). The attacker first gets a list of all file sizes and records them using /employee/report/files. Then, the attacker orders a new item for the project in question. Lastly, the attacker requests the file sizes again. The project's file size can be found by looking at the difference between the two reports. There should be one new file, and one file which increased by exactly 44 bytes between the two requests. This latter file is the project file. The attacker can take the size from the initial run, subtract 24, and then subtract the length of the project's name. Then divide that result by 44 to obtain the total number of employees who have logged hours plus the number of types of items which have been expensed. This is the answer.”

Take-Home Evaluation:

Utah identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.2 BattleBoats 1

A.5.7.2.2.1 Question 006 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“The key command in the expect script of player 1 (the attacker) is: shoot 3 100 45E25 90 The third argument to the shoot command determines the vertical angle of the shot. When player 1 executes the shoot command *45E25* is sent as a string to player 2 who calculates what the shot hits. The relevant code is in StrikeLocator.java. First the string representation is parsed into a double (a very large double) and then the app attempts to reduce the angle to less than 360 in the method:

```
public double toRadians(double degree) {
   while (degree < 0.0) {
      degree += 360.0;
   }
   while (degree >= 360.0) {
      degree -= 360.0;
   }
   return degree * 0.017453292519943295;
}
```

The problem is the second while loop, because the double is so large subtracting 360 does not change the value of degree, so the app is caught in an infinite loop. Because the calculation is done on Player 2's side when Player 2 attempts to run any command (in particular the benign request "help") the app will hang until toRadian finishes. Although the attack was discovered while trying to understand the app by playing a game, Jaam was very useful in determining the cause of the attack. Shot commands with incorrect parameters produce a particular string message. Finding the string in the code was very simple in the constructor of StrikeLocator. The question was where to look, to make sure we did not miss the cause we chose to look in the "parent" of StrikeLocator in the graph which is Competition.fetchStrikeSquares.

Using Jaam we reduced the graph to only the relevant nodes (see `question_006/MethodsInvolved.png`):

- The highest highlighted node is fetchStrikeSquares

- Above it are the messaging handling calls which eventually call fetchStrikeSquares (eg. handleRecievedMessage)

- Below it are the two branches

- Left branch is StrikeLocator's constructor which eventually calls toRadian whose leaves are the two loops, the highlighted node is the loop which causes the attack.

- Right branch is the code which calculates the trajectory and which pixels, ships etc, are hit

By exploring the left branch first, it was simple to find the cause of the attack.”

Take-Home Evaluation:

The challenge program contains an intended vulnerability in that an attacker can submit an input that causes the projectile to never hit the ground. Utah identified an unintended vulnerability in how inputs are processed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.3 BattleBoats 2

A.5.7.2.3.1 Question 010 (SC Time, Intended Vulnerable, Live: Yes)

Live-Collaborative Response:

“For each shoot OceanPanel.PullOceanSquares checks a condition (1) that if the hit square is not on the OceanPanel, then it calls Competition.isOver() (2), which has a 30ms wait on it. Since the other lines outside of the condition doesn't take long, the difference between the running times would be tangible. […] By monitoring the running time for each shot and looking at the time difference, the attacker can find out if the shot was on the OceanPanel or RadarPanel. […]”

Live-Collaborative Evaluation:

Utah identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.3.2 Question 021 (AC Space, Intended Vulnerable, Live: Yes)

Live-Collaborative Response:

“We collaborated with the GrammaTech team. Their analysis using their tool and ours both pointed to the HitLocator constructor and computeHitTime as the most probable attack channels. The constructor receives the shot configuration as strings to the appropriate parameters, and then passes them directly in to the constructor of Big Decimal objects. We initially thought that perhaps large values would affect the program, but the values are clamped to certain ranges (for example velocity is between 0 and 150). We moved on to look at HitLocator.computeHitTime and noticed a suspicious HashMap that uses BigDecimal as Key's. The hash map is used as a termination condition on a loop which is doing Newtonian Method Approximation to calculate the HitTime. We still think that is a possible attack, only needing to generate an input which takes a large (at least 26) number of iterations to occur.

Regardless we also looked at BigDecimal calculations that occur in the loop first. HitLocator.computeHitTime calls the advance function, which does division on two BigDecimals (height and verticalRate): […] Crucial is the last add in both outcome calculations. In both cases the add is missing the "HitLocator.mc" parameter. As a result, the calculation returns a BigDecimal with the same precision as the two inputs. The two outcomes are then passed to the height.divide(verticalRate, HitLocator.mc). The way BigDecimal works is that it first calculates the result with perfect precision and then applies the rounding according to the math context (HitLocator.mc). The attacker can make the calculation take very long and use a lot of space by inserting very small (due to clamping this is the easiest way) numbers in the height and initial velocity parameters.

But this is still not enough. For exceeding the limit, the loop needs to be run multiple time. As we investigated the code, we found out the algorithm in computeHitTime is trying to estimate the hit time. Its first guess is 10 seconds and because 10 seconds is always enough for the ball to

pass the ground (Considering the max height = 15, and max velocity = 350), then in the next rounds of the loop it reduces the estimated time by "h/v". The loop does this until h becomes less than epsilon (1E-10,000) or the time difference becomes less than epsilon, which means it's almost a hit.

The algorithm's next guess is where the slope of ball's movement hits the ground line (so always the ball is below the ground line) and the algorithm tries to converge. The trick is to eliminate the effect of convergence by having small values for "h" and "v". In this case they have almost no effect on the formulas for computing "h/v" which stays approximately "t/2" in each round. So, the attacker by setting "v" = 5E-10001 and "h" = 1E-10001 can make the loop to run more than 100 times. […]"

Live-Collaborative Evaluation:

Utah identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.4 Searchable Blog

A.5.7.2.4.1 Question 018 (AC Space, Intended Not Vulnerable, Take-home: No)

Take-Home Response:

"The input budget is 4MB and the output budget is 2GB. We operated under the hypothesis that getting from 4MB to 2GB requires quadratic behavior in some data structure.

Analysis of the code revealed that the only data structure likely to exhibit such behavior was com.bbn.RankingService.A. Further analysis revealed that this matrix grew only when com.bbn.MatrixRoutines.concatenateColumn is called. However, com.bbn.MatrixRoutines.concatenateColumn grows the matrix only when the number of rows in com.bbn.RankingService.A is different from the number of elements in com.bbn.RankingService.files. However, com.bbn.RankingService.files starts with the same number of elements as there are rows com.bbn.RankingService.A and never changes. Thus, these two numbers start the same and will stay the same. Thus com.bbn.RankingService.A does not grow and does not exhibit the quadratic behavior needed by this attack."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. The challenge does not necessarily need quadratic behavior on data structures for a 4 MB input budget to result in 2 GB of data consumption. A single input could trigger non-termination of a single while loop for example.

Post-Take-Home Analysis Ruling: Correct

A.5.7.2.4.2 Question 027 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"For a timing attack on searchable_blog, run the example_upload.sh script with the following as your example file in our tests:

```
<html>
 <head>
  <meta charset="UTF-8"/>
```

```
  <title>title</title>
 </head>
 <body>
  <a href="/userblog.html"></a>
 </body>
</html>
```

The attack works by being a blog entry that contains a reference to itself."

Take-Home Evaluation:

Utah identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.5 STAC SQL

A.5.7.2.5.1 Question 013 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"stacsql Huffman-encodes database fields based on the raw bytes of the inserted value. When a field is updated, the Huffman table is recalculated /except/ in the case of a BLOB field where it is updated. Recall that a Huffman table can be represented as a binary tree and the encoding of a datum obtained by the path from the root to that datum's node. When `stacsql` updates a Huffman table (as opposed to recalculating it), it simply makes the existing table a child of a new tree (with newly-appearing bytes appearing in the other child). The sequence generated by this script ensures that the constructed Huffman table is essentially a linked list. It then updates the fields to a large byte sequence of the most expensive-to-encode bytes."

Take-Home Evaluation:

Utah identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.6 Stegosaurus

A.5.7.2.6.1 Question 003 (SC Time, Intended Vulnerable, Take-home: Yes, Live: No)

Take-Home Response:

"Our initial investigation revealed a possible side channel but we have not been able to complete the attack. Stegger.hide contains a loop whose number of iterations depends on `value` raised to the 80th power. Furthermore, `value` is 0 when the bit to be encoded is 0 and non-zero otherwise. This effects the runtime of that loop, which we verified by instrumenting the code. The progress of encryption is leaked via `/progress.data?id=$thread_id`, and this could be used to determine how quickly encryption in proceeding. The remaining difficulty is whether this leak is fine grained enough to detect the fluctuations in time caused by `value`. We are still investigating this."

Take-Home Evaluation:

Utah identified most of the components of the intended vulnerability; however, the team missed the race condition that allows for a segmented side channel exploit. This intended vulnerability was discovered to be of insufficient strength.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

The team answered "Yes" during the take-home but changed their response during the live.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.7.2.7 Tawa FS

A.5.7.2.7.1 Question 020 (AC Space, Intended Vulnerable, Live: Yes)

Live-Collaborative Response:

"There is a `for` loop in the method `Filesystem#defragment`. It loops on all the inodes and try to move the inodes of the fragmented file(s). When an inode can't be clean moved, the `FileTable#backupInode` will be called. The attackable part is the `do ... while` loop in this `FileTable#backupInode` method. If we can override the file during the time it is being defragmented, the loop will never stop, and keep writing data to the disk, the <userName>.fs file."

Live-Collaborative Evaluation:

Utah identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8 Vanderbilt University

A.5.8.1 Vanderbilt Overview

Vanderbilt answered 29 questions with a 59% accuracy during the take-home engagement. In the live engagement the team's accuracy increased to 93%, the 5th highest percentage point increase

of 28 percentage-points. Vanderbilt had the 5th highest take-home accuracy and the 2nd highest live-collaborative accuracy. Vanderbilt had the most difficulty during the take-home engagement with AC questions with a combined 43% accuracy on these questions. Additionally, the team's "No" accuracy was 33%. Vanderbilt's take-home responses indicated that some of the AC analysis was done only with JAnalyzer without their symbolic complexity analysis tool. This was observed on questions 021 and 026 in BattleBoats, BraidIt Question 22, and StuffTracker Question 30. It is unclear if JAnalyzer offered sufficient capabilities for these questions or if these questions could not be analyzed symbolically for another reason.

During the take-home, the team identified the first unintended not vulnerable challenge in the STAC engagement. SimpleVote Question 25 contained an intended vulnerability, but the team accurately determined the intended vulnerability but noted that a loss of information during one of the key-processing stages resulted in a non-vulnerable application. Further analysis confirmed Vanderbilt's findings and during the collaborative engagement all teams correctly responded that the application was not vulnerable.

During the collaborative engagement, the team's analysis of noise for side channel questions was noted by other teams as very helpful. Finally, Poker Question 004 presented an interesting collaboration opportunity as GrammaTech and Colorado identified a weak side channel and ruled correctly that it was not sufficiently strong. Vanderbilt and Iowa identified potential side channels but ruled them sufficiently strong. The decision point for the presence or absence of a vulnerability was due to whether the random seed used to pad packets could be determined. As the result of collaborations, all teams correctly concluded that there was no way to determine the random seed for the benign user and predict the padding sequences, as separate *Random* objects are used in the benign user and attack threads and the challenge question limits the attacker to a single game with the benign user. This was a highlight of what was expected from the collaborative engagement.

A.5.8.1.1 Take-Home Engagement Scores

Table A-109: Engagenemt 5 Vanderbilt Take Home Engagement Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	7	78	6	67
SC Time	6	4	67	4	67
SC Space/ Time	1	1	100	1	100
AC in Space	7	3	43	3	43
AC in Time	6	3	50	3	50
Total	**29**	**18**	**62**	**17**	**59**

Table A-110: Engagement 5 Vanderbilt Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	23	13	**57**	12	**52**
Not Vulnerable	6	5	**83**	5	**83**
Yes Answer	14	13	**93**	12	**86**
No Answer	15	5	**33**	5	**33**

A.5.8.1.2 Live Engagement Scores

Table A-111: Engagement 5 Vanderbilt Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	9	8	89
SC Time	6	5	83
SC Space/ Time	1	1	100
AC in Space	7	7	100
AC in Time	6	6	100
Total	**29**	**27**	**93**

Table A-112: Engagement 5 Vanderbilt Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	23	21	**91**
Not Vulnerable	6	6	**100**
Yes Answer	22	21	**95**
No Answer	7	6	**86**

A.5.8.2 Vanderbilt Responses

A.5.8.2.1 Accounting Wizard

A.5.8.2.1.1 Question 011 (SC Time, Unintended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"There are two different types of things that can be expensed in accounting wizard: hours and items. Of these, one type (hours) are considered required. This means that all the sum of the cost of all hours MUST fit within the project expenditure limit. An attempt to add more hours than will fit will result in an error and the hours will not be added. The second type (Items) are not required. More items can be expensed than will fit in the expenditure limit. When a project manager asks for an expense report for that particular project, not all expensed items will be included but instead some subset that fits within the expenditure limit will be.

We first distinguish two different properties. One is the expenditure limit which we wish to learn. The other is the remaining budget left once the total cost of all the required items has been subtracted from the expenditure limit. We call this property the non-required expenditure (NRE). Recalling that an attempt to add more hours than is possible based on the expenditure limit and hours already expensed leads to an error message and failure to update, a binary search style attack would allow the attacker to determine the NRE value. However, this only means that the attacker has uncovered how much remains in the expenditure after the required items have been subtracted. We have no prior on the number of hours already expensed by different users. Additionally, while it is possible to delete an item (even if you don't know the item name, cost or quantity), it is not possible to delete hours except your own meaning that the attacker cannot simply remove hours in order to discover the true expenditure limit.

Thus, an attacker must either determine the total cost of the already expensed hours or the expenditure limit directly. Recalling that the project manager is checking all aspects of the project, we used Profit to run a series of experiments to determine whether the time taken to perform any of the project manager's tasks is correlated with either the expenditure limit or the total cost of hours. We found there to be no correlation. Additionally, we considered an employee's active options such as updating their hours. We experimentally confirmed that the time taken to perform these actions is again not proportional to either the cost of the hours or the expenditure limit nor is the time of the project manager's checkup following these actions."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. However, Iowa and Northeastern identified an unintended vulnerability during the take-home where under certain conditions, an attacker-controlled input of hours is directly compared against the secret. This initially weak side channel can be amplified using either the intended AC Space vulnerability or by using the remaining input budget.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Through collaborating with members from Iowa and GammaTech, we developed a side channel exploit strong enough to obtain the secret information. This side channel exploit was constructed through a collaboration with members from the Iowa team who realized the initial side-channel vulnerability and the GrammaTech team who realized how it might be augmented. We were then able to confirm their suspicions empirically. A user of accounting wizard has the functionality to add his or her own hours to a project. In the case when no hours for this user have been previously charged to the project, this results in a log file being created. During this process, whether or not the number of hours the user is adding is larger than the expenditure limit is checked (com.bbn.accounting.objects.FileStore.newHours). An exception is thrown if this is the case and the log file is not created. We noted that this may introduce a timing side channel as to whether or not the number of hours added is larger than the expenditure limit. Since a user can remove their hours (and their corresponding log file) from the project by updating their hours with the value of 0, this would mean that an attacker could iteratively exploit this side channel to create a binary search style attack to determine the expenditure limit. However, we determined experimentally that though the timing side channel is present, it is not completely reliable. […]

First, we enable logging by changing the language to a non-existent one. Second, we add a number of budget lines to the project through ordering different, inexpensive items. Now, when

the file is created for the new Hours object, a pointer to it is added to the file for the project. Because logging is enabled, the log file of each child of the project (including all the items added by the attacker) is now visited and its id printed. Depending on the number of budget lines in the project (which we can increase by ordering more items), this procedure may be expensive and thus augment the cost differential between the paths taken when the hours to be added are above or below the expenditure limit. We used Profit to demonstrate this augmentation […]

Above, we observe the timing differences between when the hours to be added are above versus below the expenditure for a different number of items in the project. We also demonstrate that Profit was able to automatically detect that this information is fully leaked (according to the experiments we ran)"

Live-Collaborative Evaluation:

There was a mistake in packaging this challenge program: the application came provided with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects' contents or states.

The unintended vulnerability discovered by Northeastern and Iowa and amplified during collaboration was confirmed.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We used JAnalyzer to get an initial understanding of the application. Accounting Wizard stores two files: (1) the ".store" folder as database, and (2) the ".accounting.log" log file. We investigated all functionality, which is accessible to an employee: change language, login, logout, wdp, order item, delete item, update hours, get hours and report files. Based on our observations no combination of commands within the input budget of 50kB can lead to a logical size of more than 1MB in these two file possibilities."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attacker can trigger verbose logging by causing a custom exception to be thrown.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"As we already reported, Accounting Wizard stores kind of files: (1) the ".store" folder as database, and (2) the ".accounting.log" log file. We have been aware of several log messages for operations like adding an order, but in our prior analysis we concluded that there is no combination of operations that trigger a space vulnerability for the given budget. The team from Utah pointed us to the fact, that if the employee attempts to change the logging language to an unsupported language, then Accounting Wizard switches its logging level to "trace" mode, which produces significantly more log messages. Afterwards, the log file logs a lot of information for every order command. […]

Under the assumption that there is an existent project NEWPROJECT that the employee can use to order items (the organizer confirmed that this is a valid assumption). The Item ordered can have arbitrary, but valid, values.

The tools used are still: JAnalyzer, Debugger. They pointed to the actual suspicious methods, but since the vulnerability is only visible in a certain log level, and reaching this log level is very unlikely, we did not see this in our first analysis.

In general we could use our fuzzer KelinciWCA to generate a sequence of operations that automatically retrieve an exploit, but since a simple sequence of order items operations is enough, the execution of the fuzzer seems unnecessary."

Live-Collaborative Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.1.3 Question 023 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"If the total amount of ordered items exceeds the specified project budget, then the Expenditure Report feature in AccountingWizard will try to maximize the number of items included in the report within the budget. Therefore, it uses an algorithmic solution of the packing problem and it will try to find the best combination of ordered items. This of course can have a combinatorial complexity. In order to find an input that exceeds the resource usage limit, we applied our tool KelinciWCA on the method: BudgetSolver.getAffordableLines(input, adjustedExpenditureLimit); with the attached driver (Driver.java).

Each byte in the input file is read as the cost of an item, and the variable N limits the number of items. It is relatively easy to find an input with a lot of items, e.g., N=100 by simply submit items with cost from 1 to 100, with a budget of 1000, that exceeds the resource usage limit. We used KelinciWCA to minimize N and still exceed the resource usage limit. Therefore, we conducted the following experiments: the budget is fixed to 1000, starting from N=38 we decreased N until KelinciWCA cannot longer generate a working (usable for this attack) input within 5 hours. In order to get results as soon as possible we configured KelinciWCA with a timeout equals to the resource usage limit (500 sec = 8.33 min)."

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“The database for accounting wizard is persistent. This persistence is managed through hidden files in the server's .store directory. This directory contains a number of XXXXXXXXXX_chunk files, where XXXXXXXXXX is a 40-char unique ID. These chunk files represent the persistent data, in the shape of a tree. While the content of these chunks are inaccessible to an employee, it is possible for an employee to discover the sizes of the hidden files through the "/report/files" GET request. We will show that this information allows an employee to discover the number of budget lines of any project they are working on. There are three different types of chunks. There is a root node, a node for each project, and a node for each budget line (could be hours or items).

The size of the root node in bytes is 22 bytes + 44 NP, where NP is the number of projects in accounting wizard. The size of a project node in bytes is 12 bytes + |projectName| + 12 bytes + 44 NL bytes. The size of a budget line node is 12 bytes + |lineName| + 4 bytes + |projectName| + 18 bytes, where lineName is the name of the item if it is an item, and the name of the person if it is labor. Importantly, if a budget line is deleted, its node is removed from the store and any pointers to that node are deleted, meaning the size of its project node drops appropriately.

Since the employee works on the project, we assume that he or she knows the project's name (this is required for actions such as expensing hours or items). Thus in order to determine the number of budget lines for any given project, an employee need only learn which node size is that of the project node in the tree. This can be done by first obtaining the sizes of the files in the store, expensing an item, and then again obtaining the sizes. Exactly one budget line node will have been added and one size will have changed. The size of the added budget node is predictable, so it is possible to determine which node size was changed by comparing the current sizes with those before the expense.

Let N be this size. The number of budget lines for the project currently is then ((N - 24 - |projectName|) / 44) and the number before the additional expense is one less than the current number. In the off-chance that the item the employee added was already a budget line for the project, no additional budget line node would have been created. This makes this edge case detectable and the experiment could be run again with a different expense or an item could be removed for the same result.”

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.2 BattleBoats 1

A.5.8.2.2.1 Question 006 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“The various hashCode methods hashes the Hit object’s various fields using the standard Java runtime hashCode methods, so there is likely no vulnerability here. Likewise, the toBuilder recursion provides the auxiliary functionality necessary to print the game’s various error messages. It does not seem that it would be possible to trigger an infinite recursion here through the benign user commands. The various PLUGINArray/PLUGINObject recursions deal with classes that extend the standard Java HashMaps and ArrayLists, and use the hasNext() methods to safely iterate through these structures, so there is no opportunity for user input to trigger an infinite recursion here. Furthermore, these methods are used in parsing the player ID files that contain the network information (host name, port, and private key) that run during startup. They would not be the source of a vulnerability.

In the loops section of JAnalyzer, our tool reports several methods in the PLUGINObject and PLUGINArray classes as suspicious. However, as previously mentioned, these could not be the source of a vulnerability. JAnalyzer also reports nested loops in Competition/printOceanBoard , though after further testing there is no board configuration that would make this method exceed the provided limit of 120 seconds.

While StrikeLocator/calculateStrikeTime is a candidate for a potential exploit, the method keeps track of changes in the variable tCurrent , and breaks out of the loop if the delta value decreases for too long. A vulnerability in this method is therefore not likely. The program features functionalities to run scripts and repeat commands from the console, though these functionalities are unreachable through benign user interactions, since the program’s Stage enumeration does not list either commands in the various list constructions. It should also be noted that it is possible to make the applications console run indefinitely by having a user attempt to connect to themselves. Any subsequent command entered will run indefinitely, never returning console control back to the user. However, this is likely a vulnerability within the io.netty.channel package, and is therefore considered out of scope for this problem.”

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attacker can submit inputs such that the projectile never hits the ground causing the calculate StrikeTime algorithm to run endlessly. Vanderbilt flagged the StrikeLocator/calculateStrikeTime method via JAnalyzer, but ruled it out as potentially vulnerable. The response does not indicate that this was run against the symbolic search of the input space to establish bounds. This highlights an unsound step in Vanderbilt’s approach where understandably, the entire program cannot be analyzed symbolically, therefore an initial unsound approach is used to filter the application.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“We collaborated with Utah team, and they convinced us that there is a AC time vulnerability. We will change our answer from NO to YES. After the players are connected, a game offered and accepted, and both players have placed their boats, the attacker only has to issue the command "shoot 1 1000 10E45 10E45" where the last 2 entries are the elevation angle and board angle respectively. The angles are reduced and converted to radians using the StrikeLocator.toRadians method, which adds or subtracts 360 to the value until it falls in the range of (0 - 360). If the value is very large, such as 10E45, it will cause the loop to iterate a huge number of times, which will far exceed the 120 sec resource limit. The cost of the

command is very small (less than 100 bytes after the initial setup) and is well within the budget of 10kB.

Our JAnalyzer tool can find this loop as suspicious. It is just a single level loop, but it's an open-ended while loop on unrestricted user input that is a floating-point value, hence the loop is pretty unrestricted. The method is called when a StrikeLocator object is initialized, which occurs when Competition.fetchStrikeSquares is executed, which occurs when a SHOT_MADE message is received from the other player."

Live-Collaborative Evaluation:

Vanderbilt worked with Utah and reported the same unintended vulnerability flagged by Utah.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.2.2 Question 028 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"Battleboats has two players playing a game similar to battleship; one player is the attacker, the other is the victim. Both players have cannons which fire at their opponents ships, but they do not know where their cannon is (as the opponent places it on their board). The question asks if there is a side channel in time in which the attacker can deduce the exact coordinates of their own cannon. When a player fires their cannon, they give four parameters, rather than coordinates: height of cannon, initial velocity, elevation angle, and board angle. By playing with these four parameters, a player can shoot their cannon to any place on or off the board. However, the player does not know if their shot landed in the board (unless the shot overshoots the board by a landslide) or not: they only know if they HIT or MISSED a target.

A shot can be:

1. on the board
2. off the board, but less than 1 boardswidth away
3. off the board, but more than 1 boardswidth away

If the shot is on the board, or less than 1 boardswidth away, the attacker is sent a list of positions hit from the other player; when a shot far overshoots the board, as in case 3, the attacker is notified that the cannon shot overshot the board by too much. Unfortunately, this is not enough to discern the position of the cannon. By analyzing the # of packets sent and size of packets in all 3 cases, we can identify whether or not the size of the packet is related to the shot being on or off the board. Our analysis using the package analysis component of our tool Profit leads us to believe that there is no side channel, as the only change in packet size occurs when the attacker attempts to shoot at an (X,Y) coordinate where X and/or Y is negative. While a player can use other commands than shooting their cannon, none of them appear to leak any information about the secret. Thus, nothing in the code of the application relating nor the size of the packets sent during a battleboats game give any indication of a space side channel."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.3 BattleBoats 2

A.5.8.2.3.1 Question 010 (SC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“Battleboats has two players playing a game similar to battleship; one player is the attacker, the other is the victim. Both players have cannons which fire at their opponents ships, but they do not know where their cannon is (as the opponent places it on their board). The question asks if there is a side channel in time in which the attacker can deduce the exact coordinates of their own cannon. When a player fires their cannon, they give four parameters, rather than coordinates: height of cannon, initial velocity, elevation angle, and board angle. By playing with these four parameters, a player can shoot their cannon to any place on or off the board. While where the shot lands is obscured from the attacker by use of an auxiliary "radar" board, the computation of the trajectory and analysis of what is hit by the shot is performed on the opponent's true "ocean" board. Therefore, something in that computation might leak information about the position of the attacker's cannon.

We used our imbalance analysis tool, CoCo-Channel to analyze the locations of potential side channels in code. The method below, stac.combatboat.OceanPanel.pullOceanSquares(x,y), is called when the attacker fires a shot in order to calculate the squares the shot hit. Here, pixelX and pixelY are dependent on the secret (the position of the cannon). Our tool was able to detect that there is a potential imbalance in the amount of work done based on whether the if(!this.onOceanPanel(pixelX) || !this.onOceanPanel(pixelY)) branch is taken. Specifically, if the shot lands outside of the board but is not so long a shot that it itself is more than the length of the board, then double the amount of work is done compared to when the shot lands within the board. If this introduces a noticeable timing difference, then an attacker will be able to deduce if their shot landed within the benign user's ocean board. […]

We used the packet analysis portion of our tool Profit to confirm that the timing difference between these two cases is noticeable; see the attached graph battleboats_2_cannon_timings.png. We averaged the time from 50 shots landing out of the board, 50 shots landing in the board, which yielded a consistent timing difference of ~20 milliseconds on the reference platform. From this, an attacker can use a binary search-esque approach to exploit this channel and find the coordinates of their cannon. We begin by finding out "X" coordinate. We shoot perfectly horizontally for half the length of the board. We then use the side channel to determine if we have shot in or out of the ocean board and continue our search until we have found our "X" coordinate. We repeat this procedure to obtain the "Y" coordinate. Since the largest board size is 20, it is possible to determine the location of our cannon within the budget.”

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.3.2 Question 021 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"The various hashCode methods hashes the Hit object's various fields using the standard Java runtime hashCode methods, so there is likely no vulnerability here. Likewise, the toBuilder recursion provides the auxiliary functionality necessary to print the game's various error messages. It does not seem that it would be possible to trigger an infinite recursion here through the benign user commands. The various JACKArray/JACKObject recursions deal with classes that extend the standard Java HashMaps and ArrayLists, and use the hasNext() methods to safely iterate through these structures, so there is no opportunity for user input to trigger an infinite recursion here. Furthermore, these methods are used in parsing the player ID files that contain the network information (host name, port, and private key) that run during startup. They would not be the source of a vulnerability.

In the loops section of JAnalyzer, our tool reports several methods in the JACKObject and JACKArray classes as suspicious. However, as previously mentioned, these could not be the source of a vulnerability. JAnalyzer also reports nested loops in Competition/printOceanPanel , though after further testing of various board configurations, there is no vulnerability here. Among the methods listed in JAnalyzer's news section, of note are the following:

- Competition/observeShipSquares
- WarShips/showHitOutcomes
- HitLocator/computeHitTime

observeShipSquares is called every time a ship is set. Based on the length of the ship, the program will allocate a new Square at each (x, y) coordinate that is defined. However, given the short lengths of the ships and the input budget, triggering a vulnerability in this fashion would not be possible.

showHitOutcomes is the least likely of the three, with no explicit calls to new. The method is executed after a shot is made and checks to see if the cannon has been hit, whether the shot was on or off the board, and if the shot hit a target. This method is not a candidate for a vulnerability.

computeHitTime allocates a new HashMap and eventually begins adding elements to it once the iterations through the method's while loop reaches a threshold. However, upon each iteration, if the tCurrent key already exists within the HashMap, the loop breaks, preventing any potential exploits that aim to make this loop run and add to the HashMap an indefinite number of times. Therefore, we discard this method as potentially vulnerable. The program features functionalities to run scripts and repeat commands from the console, though these functionalities are unreachable through benign user interactions, since the program's Stage enumeration does not list either commands in the various list constructions."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The application of Newton's method to calculate time for the projectile to hit the ground can hit a slow convergence case on certain inputs, consuming both memory and time. The vulnerable computeHitTime is flagged for allocations to the HashMap but ruled out on the condition that only new tCurrent keys are added to the HashMap. The team missed potential for slow convergence filling this hashmap.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We collaborated with Colorado team, and they convinced us that there is a AC space vulnerability. We will change our answer from NO to YES. The main exploit uses following send command, which is sent for about 7 rounds. send […].

The first parameter with a small precision, in this case (1E-100), is the height of the cannon, which is used in a calculation in HitLocator's advance() method. This data is then put into a HashMap in the computeHitTime() method. The amount of times data is put into the HashMap is bounded by the amount of time the projectile is in the air, so a non-zero velocity (the second parameter of shoot) is optimal. When this is processed, the loop in the com.cyberpointllc.stac.combatboat.HitLocator.computeHitTime() method will iterate until it cannot find a better estimation. So, given the very small precision, the loop will keep adding to the HashMap trajectory. When looping many times, this will make the memory usage exceed 800MB.

Our JAnalyzer marks this loop as suspicious since it can be reached from some entry and its loop condition can be affected by the computation in the loop."

Live-Collaborative Evaluation:

Vanderbilt identified the intended vulnerability. Vanderbilt flagged the intended vulnerability as suspicious, but Colorado demonstrated the path to force continued loop iterations resulting in a vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.4 BraidIt 1

A.5.8.2.4.1 Question 002 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"The other player can apply any number of four different transformations to their chosen string. In the edge case, the opponent could choose to apply no transformation to the selected string but then the attacker would trivially be able to tell which was selected (modulo a collision in the randomly generated strings). Thus, it seems a reasonable assumption that at least one successful transformation is applied. Even so, since there may be a timing side channel that depends not on the transformation applied but on the number of the braid selected, we used Profit to quickly determine that there is no correlation between the number of the braid selected and time.

Under our assumption that at least one successful transformation must be applied, we looked into the four possible transformations to determine if the time taken for any could vary with time. Since we have no control over what the other player might choose to apply, we decided to show

that there is one transformation whose execution time does not depend on anything about the chosen string. Since the attacker has control over the 5 lengths chosen, the obvious choice was to see if the execution time varies by length. In most of the functions, we determined that its only length-dependent computation (other than an early return if the string is too long for an expand transformation) is in a StringBuilder operation to build the modified string. This is much too inexpensive for a timing difference to be observable. The only more expensive correlation with length occurs in triple_swap's iteration over the braid to find all possible triples to be switched. This too is unlikely to produce a major timing difference.

In addition to length, the attacker can also choose the number of alphabet characters (called strands) to use in the game. While the time of each successful transformation does not depend on the alphabet size, which transformations can be successfully applied does. However, this only limits which transformations can be applied."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. While it does not impact the conclusion or post-engagement evaluation, the assumption that at least one successful transformation is applied is not correct. There are inputs for which not successful actual transformations are applied and only the identity transform (i.e.) no transform can occur. There appears to be an assumption of a persistent side channel, while not present in this case a side channel could be triggerable only appearing in certain instances.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"We collaborated with the team from Colorado. They realized that there was a potentially exploitable timing side channel due to the recursion in the com.cyberpointllc.stac.plait.Plait.freeSimplify() method. The depth of the recursion is dependent on the length of the braid processed (which is the selected braid). This might be exploited as follows: The attacker is in charge of the choice of initial lengths in rounds 1 and 3. Assume the attacker chooses 2 strings of length 1 and 3 strings of length 50. Additionally, the attacker may choose the number of strands to be 3, meaning that the opponent cannot apply any transformation that changes the length of the string. Then when receiving the modified braid, the attacker can guess some braid of length 50. After the attacker makes a guess, the freeSimplify() routine is called before the winner is announced. The time that this takes leaks information about whether or not the opponent choose a braid of length 1 or of length 50. Performing this analysis in both round 1 and round 3 to determine the length of the chosen string and then using oracle queries to disambiguate between the strings of the same length allows the attacker to acquire the necessary information.

We briefly note that the freeSimplify() method is only recursive until it reaches a substring where no pattern of the form "Xx" or "xX" (we only use x's because of our choice of 3 strands) is found. However, because the strings are randomly generated, it is likely the recursion will be expensive enough in the case of the length 50 string that the timing difference will be apparent when compared to a string of length 1.

However, we continued to discuss the problem with a group involving teams from Colorado, Iowa, Two Six, and Northeastern. This side channel involves computation that takes place entirely on the client side. Since the attacker has full control over his or her own version of the

code, such a timing side channel does not reveal information that he or she cannot obtain directly. Additionally, the attacker would have to modify the client to obtain the necessary timing information (making a guess does not send any packets) and since all the interesting computation takes place client side, any adversary capable of making such a change could trivially obtain the information. Because of this, we are not sure if this side channel is in the spirit of the program as it provides us information about client-side behavior which we already have full observability over. Nevertheless, it is definitely a side channel in time that leaks information about the braid chosen by the other player."

Live-Collaborative Evaluation:

Vanderbilt in collaboration with other teams correctly concluded that there was not an intended vulnerability. To offer clarity on the potential side channel identified, the operational definition and reference platform for STAC establishes the role of Attacker, Benign User, and Application Under Attack. The application under attack for remote applications is always the one at the remote endpoint on the ServerNuc. As the attacker's own application instance or client interface are part of the attacker. Therefore, the attacker can view all of the data available unless it is otherwise obscured, making the presence of a side channel not relevant for these cases.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.4.2 Question 007 (SC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"There are four transformations the benign opponent can apply to the braid they select. Of these four, two are length preserving (swap and triple_swap) and two are length altering (expand3 and expand5) which increase the length of a given braid by 2 or 4, respectively, up until the length limit of 50. The two length altering transformations are applied at a specific index of the braid (e.g., the command "expand5 3" means replace the character at index 3 by 5 characters.) However, these two transformations cannot be applied at any arbitrary index. expand5 cannot be applied to the characters 'z' or 'Z'. There are four characters (two lowercase characters and their uppercase equivalents) that expand3 cannot be applied to, and which four characters these are depends on the number of allowed characters in the braid.

The number of characters allowed in the braid is called the number of strands in the braid. This parameter is fixed across all three rounds and chosen by the player offering a game which is the attacker in our case. The allowable values for the number of strands are 3-27 which control the number of characters allowed in the braid. If the attacker uses a parameter of '3', then the only characters allowed in a braid are 'z' and 'Z'. Because of this choice, neither expand3 nor expand5 will be applicable at any index of any of the five braids. This means that the opponent cannot change the length of the selected braid regardless of which one they choose. Therefore, if the attacker chooses five distinct lengths to send to the opponent, then the braid sent back by the opponent will have one of those lengths and will be a modification of the braid whose length it matches. This strategy will work in both round1 and round3 allowing the attacker to correctly determine which braid was chosen and modified in both rounds.

To solve this problem, we applied our SPF analysis to the functions growOneToThree and growOneToFive. We augmented these functions with constraints on the characters possible at the given index based on the chosen number of strands. We experimented with different number of strands and learned that: when the number of strands is set to 3, then there is no path through

either growOneToThree or growOneToFive that reaches a "return true" statement and that when the number of strands is set to 4, there is no path through growOneToThree that reaches a "return true" statement. This means that these functions cannot be applied in these cases."

Take-Home Evaluation:

The challenge contains an intended vulnerability where an attacker can trigger or simply observe packet sequences and combine this with knowledge of the compression scheme to leak the secret. The side channel discovered by Vanderbilt is one that was realized following the start of the engagement.

Post-Take-Home Analysis Ruling: Correct

A.5.8.2.4.3 Question 017 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"This question involves attempting an attack by one player that will cause the program to create a file of size > 25 MB. The only file created by the program is the log output file, so we are interested in looking at how it can be led to produce the desired file size. Output files are produced by the library slf4j which uses a log4j.properties file to specify how its output is produced. This one is set to place the log output files in the location staclog/stac.log and limits the file size to 10 MB, but is setup as a RollingFileAppender, meaning when a file reaches the max limit it is renamed and a new file created for up to MaxBackupIndex times, which is set to 3. This means the logger output is allowed to produce up to 30 MB of output, which will exceed the resource limit above. So we will look at ways in which the logging output can produce these large quantities of output. It appears that the only output levels used are INFO and DEBUG, with DEBUG disabled, so all logger output is generated by the logger.info calls. None of these individual calls seems capable of producing much output (around 70 - 150 bytes per call), so the only way to do this would be through looping or recursion of the calls on the order of around 100K times. Note that on this version the commands Repeat and Script , which would be helpful in producing recursive calls, are both disabled.

This seems to be the most only likely possibility of inducing a large output file to be produced.The conditions are:

- isSimplified must return false, meaning that the braid must on each iteration have a matching upper and lower case version of the lowest character value in the braid. This is somewhat muddled in that if the value of bottom is < this lowest value, it will return true. However, this does not affect the outcome of the test.
- obtainEnsuingPortion finds a portion (defined above) in the braid in which all the following are true (returns null if no portion found):
    - if the 1st char is 'A' or 'a and the portion length is <= 26 characters (this is only pertinent if the maximum number of fibers (27) is used)
    - if the 1st char is not 'A' or 'a' and the portion does not contain either case of the prevchar value
    - if the 1st char is not 'Z' or 'z' and the portion does not contain both cases of the nextchar value
- simplifyOnce does nothing (including output a message to the log file) if the portion described above is not found. Otherwise, it creates a replacement for the portion by

removing the 1st and last char of the portion and performing the following (all references are to the 1st character):

- if a character matches the inverse of the nextchar it appends 2 characters: the inverse and the nextchar (e.g. if 1st char is 'X' then all cases of the value 'y' will be replaced with "yxY")
- if a character matches the nextchar it prepends 2 characters: the inverse of the nextchar and the 1st value (e.g. if 1st char is 'X' then all cases of the value 'Y' will be replaced with "yXY")

This has the effect of potentially creating more portions in the braid by adding in more copies of a prevchar and its inverse . Note that the resulting braid length may either be reduced by 2 (if there are no copies of nextchar of either case in the portion), kept the same (if 1 copy is found), or increased by N * 2, where N is the number of nextchar values found -1. Since the maximum length of the braid and therefore the portion is 50, the number of occurrences can be as much as 48, meaning the length could increase by 94 to a length of 144. This could be a way to cause the process to repeat for a large number of times.

- freeSimplify this simply goes through the braid and eliminates all matching pair values. If a pair is removed the method will recurse until there are no more consecutive matching pairs in the braid.

Note that this method outputs a message on each iteration it performs, but will remove a minimum of 2 characters on each pass. This problem does seems likely to have a vulnerability due to the simplifyCompletely method. It appears that there are multiple braid patterns such as: " abbbbbbbbbbbbbbbbbbbbbbbbA " that cause the simplification algorithm to run continuously and will produce 3 files: stac.log.1 , stac.log.2 , and stac.log.3 , each having a file size of 10.5 MB to be produced, yielding a total of 31.5 MB, which does exceed the limit.

The 2nd part of this is how to take a random braid and using the given commands to produce this pattern. We need to determine if an input can be coerced that will cause simplifyOnce to keep extending the braid while freeSimplify removes pairs and creates log messages. One difficulty though is that the initial braid is randomly generated and the attacker only has the 4 different modifications (expand3, expand5, swap and triple_swap) to work with to produce the required braid pattern.

The simplest braid pattern seems to be yzY , since it contains no matching pairs initially so freeSimplify does not modify it and the only portion in the braid is the entire braid yzY, which does contain the next character value within it, so it does not reduce down. This can be produced with a braid containing the minimum number of fibers, so it doesn't matter what the opponent chooses as the number of fibers. Also this pattern works if the character values are all reduce by the same amount (e.g. abA also works) and the intervening characters can be of any length as long as the values are all the same and are equal to the nextchar of the 1st character in the braid (either case). Also, since the freeSimplify method simply removes all matched pairs there can also be any number of these inserted in the braid and they will be reduced down to this same pattern.

Now all we need to do is take an arbitrary braid pattern and be able to modify it with the 4 commands to produce this pattern. The most useful commands are expand5 , which will add 4 characters to the braid and swap that will swap 2 sequential characters in the braid. If the initial number of fibers is set to the minimum of 3, the initial random braids will all consist of only the

chars 'y', 'z', 'Y' and 'Z'. The expand5 command can only be used on the 'y' and 'Y' characters and would transform them into "zyzYZ" and "ZYZyz" correspondingly. By then doing a swap on the 1st char (in case 1) would result in "yzzYZ" and then again on the 4th would yield "yzzZY" which would reduce down to "yzY" and would indeed exceed the output file size limit. This would be produced from a single 'y' character in the braid. Any 'z' characters could always be "swapped" to move the character into the center so that a "yzzz...zzzY" pattern is then formed, which also exceeds the file limit. If the number of fibers is set to 3, it should be possible to issue the appropriate commands to generate one of the patterns described that will cause the maximum file limit to be exceeded. Since the commands may be used as many times as needed without affecting the budget the question is can a random braid of multiple "strands" (different alphabetic characters) eventually be transformed into a braid of the type mentioned. Yes, as long as you do not exceed the maximum braid length of 50 characters. For instance, starting with the pattern "yzzXzY" you can remove the "x" character by first performing the expand5 on it, making the result "yzzYXYxyzY" now by performing a swap on character 5 (with 6) you get "yzzYXxYyZY". This pattern (when the opponent makes his guess which runs the simplify operation) will remove the matched pairs and reduce to "yzzYZY". Although this pattern isn't of the correct form you will note that it now consists of only the characters { y, z } when it started with { x, y, z }."

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.5 BraidIt 2

A.5.8.2.5.1 Question 001 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"There are four transformations the benign opponent can apply to the braid they select. Of these four, two are length preserving (swap and triple_swap) and two are length altering (expand3 and expand5) which increase the length of a given braid by 2 or 4 respectively up until the length limit of 50. The two length altering transformations are applied at a specific index of the braid (e.g., the command "expand5 3" means replace the character at index 3 by 5 characters.) However, these two transformations cannot be applied at any arbitrary index. expand5 cannot be applied to the characters 'z' or 'Z'. There are four characters (two lowercase characters and their uppercase equivalents) that expand3 cannot be applied to, and which four characters these are depends on the number of allowed characters in the braid.

The number of characters allowed in the braid is called the number of strands in the braid. This parameter is fixed across all three rounds and chosen by the player offering a game which is the attacker in our case. The allowable values for the number of strands are 4-27 which control the

number of characters allowed in the braid. If the attacker uses a parameter of '4', then the only characters allowed in the braid are 'z', 'Z', 'y', 'Y'. Because of this choice, expand3 will not be applicable at any index of any of the five braids. However, expand5 will be applicable to those indices whose character is 'y' or 'Y'. This means that the opponent can only change the length of the selected braid by a multiple of 4. Therefore, if the attacker chooses five lengths such that only two are equivalent modulo 4, then the braid sent back by the opponent will match at most two of the chosen lengths modulo 4. This strategy will work in both round1 and round3, allowing the attacker to narrow the search space to at most 4 pairs of braids (2 from each round). Using the oracle queries, the attacker can determine which braids were chosen in round 1 and round 3 within the budget.

To solve this problem, we applied our SPF analysis to the functions growOneToThree and growOneToFive. We augmented these functions with constraints on the characters possible at the given index based on the chosen number of strands. We experimented with different number of strands and learned that: when the number of strands is set to 3, then there is no path through either growOneToThree or growOneToFive that reaches a "return true" statement and that when the number of strands is set to 4, there is no path through growOneToThree that reaches a "return true" statement. This means that these functions cannot be applied in these cases."

Take-Home Evaluation:

Vanderbilt's response indicates that they miscalculated the operation budget of their exploit. In remote applications, active requests are application requests sent to the remote peer under attack. There are 10 active operations required to play the game, leaving only 2 Oracle queries.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.8.2.5.2 Question 022 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"[…] The commands to run Display.runScript and RepeatCommand.execute are disabled in this program, which leaves only Weave.reduceCompletely and Weave.makeRandomModification as possibilities.

Weave.reduceCompletely is executed when the command make_guess runs. This performs the functions freeReduce followed by reduceOnce and freeReduce again continuously as long as isNormalized returns false. This can potentially be a continuous loop if this function can be forced to always return a false. The function returns false if any of the braid elements (ASCII upper and lowercase alpha chars only) contains a character that is the inverse of either lowest or base . The value of lowest will be set to the lowest value in the braid (ignoring case types). The value of base is a little more complicated. It will be the lesser of either lower or 'z' + 1 - penalty , where penalty is a value from 0 to 26 that is set by the method calculatePenalty . This method, in turn, creates a mapping of each character in the braid with the number of occurrences for each

character. If the total number of different characters in the braid is < 26, the penalty value is 0 so base is 'z' + 1. Otherwise, it returns the difference between the most repetitions and the least repetitions. Since the braid has a max length of 52 and at least 26 chars are defined, the smallest min value will be 1 and the largest max value will be 27, for a max returned value of 26, which would result in base value of 'a'. There may be an opportunity here if a braid pattern can be induced in which the braid consistently contains an occurrence of an upper and lower case version of the lowest char value in the braid after every iteration of reduceOnce and freeReduce . reduceOnce will look for a portion of the braid that start and end with the same char of the opposite case and will replace it with its contents potentially modified. The modification adds in 2 chars for every char found in the range that alphabetically follows the characters removed. This operation can reduce the braid length by 2, but can also either leave it the same size or even increase its length by 2 far each special character found. freeReduce then iteratively searches for 2 sequential characters that have the same value but are reversed in case and removes them from the braid. So is it possible for reduceOnce to modify a string that contains an upper and lowercase version of a character and replace them with a string that contains the same or lower value upper and lower case characters that are not sequential? Seems unlikely, since it will replace a value with upper and lower values that are larger than the original, not smaller.

Weave.makeRandomModification is executed when the command modify_random runs. This will randomly select one of the operations { expand5, expand3, swap, triple_swap } to perform on the braid (a random value is used for the index value for expand5 and expand3). Up to 3 attempts will be made if the process fails for the braid. If any of these operations consume much time this could be used to increase that duration by making the first 2 operations fail, but since the probability of an operation failing is based on a random value this does not seem to be a way of consuming a lot of time. Also, none of these operations can be induced to delay for an appreciable amount of time. […]”

Take-Home Evaluation:

The challenge contains an intended vulnerability. Certain inputs can cause the method that determines when braids are reduced to always return false leading to a non-terminating loop. The analysis by Vanderbilt indicates manual analysis of flagged components from JAnalyzer output rather than an automated analysis of conditions under which certain loops may fail to terminate.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Colorado pointed us to the Weave.isEquivalent() method. They used their fuzzer to generate string values (s1,s2) that trigger a worst-case behavior in this method. In order to use these string combinations it is necessary to adjust the client code of the attacker in the classes WeaveItDispatcher and SelectWeaveCommand. For WeaveItDispatcher, instead of using random generated strings, they used the s1 for all braids. For SelectWeaveCommand, instead of actually doing the modifications of the player, just return their string s2. This means that at the end, the isEquivalent() method always uses these two fixed strings that lead to the worst-case behavior (or at least to a long running execution).

Our previous answer assumed not to change the code of the attacker, but only use the existent commands of the game to exceed the budget. In previous answer we described that our tool (JAnalyzer) showed the suspiciousness of the methods Weave.reduceCompletely() and Weave.freeReduce() (which are also called by the isEquivalent() method), but we didn’t find a

concrete time vulnerability. We also did not apply our tool KelinciWCA, a fuzzer to retrieve inputs for a worst-case behavior. With the help of Colorado, we know where to start our fuzzer to retrieve a similar worst-case input. In the limited time during the collaboration engagement, we were able to retrieve following string values […]

Using these values leads to an significant slowdown of the isEquivalent() method. Unfortunately, we were not able show during the collaboration engagement that our values lead to a working exploit, and we believe that in order to retrieve an actual exploit, we would have to run our fuzzer longer. Please find attached the driver for our fuzzer: "Driver-for-Question22.java". But since our previous analysis pointed us on the same methods, although we did not generate a concrete exploit, we believe that it is possible to find these concrete values to trigger an execution that exceeds the resource usage limit of 300s."

Live-Collaborative Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.6 IBASys

*A.5.8.2.6.1 Question 015 (SC Space, Intended Vulnerable, Take-home: Yes, Live: *Yes)*

Take-Home Response:

"We used JAnalyzer to get an initial understanding of the problem. We analyzed IBASysServer.java. In method init(), an extra error message is sent, in case the result from the image matcher was success (but some error occurs), together with the correct token. This indicates a potential side channel vulnerability (the extra message sent back to the client would reveal that the token sent back by the server is the correct one).

Using similar analysis (using SPF) as for Q029, we determined that padding would trigger an error. We then similarly modified the client to add extra padding to the sent messages, to trigger an error on the server side. We can demonstrate the space side channel on the provided incorrect image and the modified client code attached. We used Prospect to validate the attack. The client needs to be run twice in the same session. The second response from the server reveals the correct token. This attack allows one to get the correct token with the wrong image."

Take-Home Evaluation:

The challenge contains an intended vulnerability that leaks the passcode string bit-by-bit. This can then be used to construct a valid image that will be accepted by the application. Vanderbilt identified an unintended but out-of-scope vulnerability similar to that identified by Two Six Labs. This vulnerability is out-of-scope in that while the conditions to trigger the vulnerability are all within the application byte code it is a direct channel that reveals the secret output that only an authenticated user should receive.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"We have identified a vulnerability and provided the exploit for obtaining the secret token of the victim which can be used to impersonate the victim to third party applications. We had some discussions on if this vulnerability is a side channel in space. We had this argument that the attacker can use the side channel in space to tell if a username and passcode are valid. The ibasys

application sends extra bytes of "error message" if they are not correct, even though, this extra knowledge has not directly been used by our exploit. Then, we had some discussion on what this vulnerability can be accounted as an impersonation. After clarification on the question we decided to change our answer to NO because the exploit does not directly use the information obtained from the side channel in space and also because the adversary obtaining the token cannot impersonate the victim to the ibasys application itself. […]

Live-Collaborative Evaluation:

As with the take-home response Vanderbilt identified an out-of-scope (i.e. not a side channel) unintended vulnerability that leaks the secret. The goal of the question was to impersonate a user from the viewpoint of outputs from the server application. This server application only provides a single output for a valid user, therefore leaking this token is equivalent to the output of the server for an authenticated user. This indicates that the token leak spans the valid set of outputs an authenticated user can receive from the server. Therefore, this response was changed from a "No" to a "Yes" and given a post-engagement evaluation of correct. The challenge does contain an intended vulnerability that leaks the secret bit-by-bit.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.6.2 Question 024 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"We used JAnalyzer to get an initial understanding of the application. We used SPF for analysis. Analysis of ImageMatcher revealed a timing side channel with "segment oracle". Analysis revealed that only a cropped image is used for authentication, with a total of 128 bytes, and from that only every other 4th byte is considered leaving 32 bytes and from that only the parity bit is considered leaving 32 bits as the actual secret. So only 32 bits need to be guessed by the attacker. This was confirmed with SPF which measured the channel capacity. This seems very likely so if the timing measurements are accurate we can easily build an attack. One needs only 31 steps to build the optimal attack with "perfect" timing info. Now the budget for the timing channel question is 3500 operations so we worked devising an experiment to observe a timing difference in 100 operations. While on server side we could observe it, on client side we ran Profit to try to observe it. We built example images (for different segments -- images attached) We ran the client with 100 operations and tried to notice a difference. The results from Profit are attached. However, the analysis did not reveal significant differences for the different segments. In conclusion, we identified a timing channel (for "segment oracles") that can potentially leak the 32 bits in the secret. However, we could not observe significant timing differences on the server side, by making measurements within the budget so the answer is NO."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.6.3 Question 029 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"We used JAnalyzer to get an initial understanding of the application. We analyzed IBASysServer.java. Method init() invokes Scalr.log(1, "ok", new Object[]{session}); when catching an exception. The log method creates an XML file in log/ directory with the content of the session. The worst case likely when submitting an image (which is sent in packets) where an error is thrown when processing each packet, so a session is saved in the XML file for each packet. Using a debugger we determined that the session is null before the invocation of method parsePacket so the method that is likely to throw an exception and generate the vulnerability is method parsePacketPart in LoginManager.java. We applied SPF to a modified version of parsePacketPart and determined an important condition corresponding to the last 5 bytes in a packet being all 0. Presumably this is due to padding. Running SPF revealed more important path conditions that are all related to checking iteratively that the symbolic bytes in the processed message are 0 or not (basically it seems that this procedure checks if all the bytes in the packet are zero to perform some calculations). This analysis revealed a vulnerability on the server side wrt padding (or simply 0 valued bytes) in the packets. We then manually modified the client to introduce extra padding in the messages that are sent to the server and created an image within budget. The extra padding determine new executions in parsePacketPart that eventually through an exception that determines the logging. The generated log file exceeds the budget. The modified client code is below and the image is attached. We validated this attack on the NUCs In the client code we modified the following method in IBASysClientParser: we added extra messages with padding which on server side triggers more image matching which gives errors which results in a log file of 1.5M."

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.7 Medpedia

A.5.8.2.7.1 Question 009 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Definitions

SCS (side-channel size) is the sum of the payloads of all server->client packets seen after the request for the desired article.

MCS (main-channel size) is the actual size of the article (HTML only).

ANL (article name length) is the number of characters in a string like A/Dream.html.

MCS of each article is deterministic

We first used a python-requests script to fetch an article many times, and confirmed that it always yields the same content. The size of the MCS coincides with the actual size of the file on the server.

No guarantee that the user will retrieve non-HTML assets

Assets served by StaticResourceController (embedded images, JavaScript files, etc.) are not padded, and thus could be very helpful to the attacker. But the example interaction scripts do not retrieve any assets besides the main HTML file. Since the example interactions constitute a spanning set of benign user actions, we assume the attacker cannot rely on these assets being retrieved.

Irretrievable articles

During our analysis we ran into a small number of article names that are in listAll but cannot be retrieved (404, not found). The traffic pattern for those cases is easy to spot, so we can tell this case from a non-broken one. The number of broken articles is much smaller than the number of allowed oracle queries, so we rule out this case.

Correlation between article request size and ANL

We suspected the size of the client request might leak information about the ANL. We set up the following Profit experiment: Launch a server; obtain the list of all articles, and retrieve a sample of randomly selected articles. Set the secret to be the article name length (ANL). Do this for N different articles, retrieving each article T times. We did this for {#articles=50, #timesEach=20} and for {#articles=256, #timesEach=4}. Profit reported that the expected feature (the size of the packet sent by the client to the server when requesting the article) fully leaked the secret. […] Using Profit's output for that particular feature, we confirmed that ANL(a) and ReqPktSize(a) had direct correlation and found out that ANL(a) = ReqPktSize(a) – 227. Since the attacker can obtain all article names and their respective ANLs offline, this vulnerability reduces any set of candidate articles to the subset of articles with a certain ANL.

SCS: up to 8 different padding values per article

We set up the following Profit experiment:

Launch a server and retrieve the article with the smallest size (1526 bytes) 256 times. We observed that:

- The distribution of the SCS was nearly uniform over 8 different sizes, all of them higher than the MCS; this seemed consistent with some sort of padding being used.
- Each time we executed this experiment again (which includes launching a new server container), the set of 8 SCS values changed. In other words, rebooting the server gives rise to a new set of 8 values.
- Besides those 8 values that kept repeating, there were a few additional values with 1 or 2 appearances. This is related to TCP connections being reestablished periodically, which can add overhead every now and then. More on this later.

Inspection of HTTP headers

Since the SCS varied but the MCS remained the same, we configured Profit to save the HTTP headers. We found that an X-Padding header is used to add a string of random characters to the response, e.g.: […] Recursively grepping the decompiled source code for X-Padding led us to 2 places where this is done, in ContentController.java and TitleSearchController.java. A quick inspection of the code suggested that the padding added to each article is a random number (between 32 and 2048) of random characters. For the smallest article, we also noted that the rest of the headers (besides the random padding string) adds a fixed overhead of 148 bytes, and something else was adding another 50 bytes.

SecureRandom with STAC algorithm

The code in ContentController and TitleSearchController seeds the SecureRandom PRNG every time an article is served, which seems unusual. But it is seeded from /dev/urandom. It seems like it still should return a different number every time. Our empirical evidence shows that it doesn't. We ran the same PRNG used by ContentController to pick the number of padding characters, and confirmed that on the server side, no more than 8 different values are generated, and that this set of values changes every time the program is restarted. A few more conclusions from this experiment:

- The distribution of the 8 numbers over the whole set of possible numbers is not necessarily uniform (e.g., the number 2038 is more likely than others).
- Once a set of 8 is chosen, the probability distribution over those 8 values is uniform.

Sessions and TCP connection keepalive/restart

During our initial experiments with the smallest article, some unusual values of SCS appeared which did not make sense given the current set of 8 numbers. By inspecting the packet traces saved by Profit, we noticed that these unusual values corresponded to cases where the TCP connection was reset and had to be re-established (i.e., beginning of a new session). The observable feature that Profit can find automatically (sum of payloads of all packets sent from server to client in the 2nd phase of the interaction) was accidentally also capturing the new TCP and TLS handshakes, which added extraneous overhead to the interaction every now and then. This can be worked around in two ways: either by using a sequence of actions known to not trigger a new TCP connection (it seems like listAll before article helps), or by refining the observable feature to something more precise, such as "total number of bytes sent from server to client after the particular packet that contains the client request". The latter can be done because the TCP+TLS handshake is a stable pattern within this application.

Discovering a correlation between SCS and MCS

We set up the following Profit experiment:

Launch a server and retrieve a subset of randomly selected articles. Set the secret to be the actual article size (MCS). Do this for N different articles, retrieving each article T times. We first did this for {#articles=20, #timesEach=100}. Besides the "size of request from client to server" feature (which, as we already showed, is a side channel for the ANL), the second highest-leaking feature reported by Profit's leakage quantification was the sum of payload sizes sent from server to client during the second phase of the interaction (article request and response).

For this experiment, Profit reported that said observable was fully leaking the secret:

Number of secrets: 20

A-priori Entropy: 4.32192809489 bits

A-posteriori Entropy: 0.0 bits

Leakage Entropy (a-priori - a-posteriori): 4.32192809489 bits

This suggested that the observable was a good one, but the result came from a narrow sample (only 20 different articles). When we inspected it, we found no collisions in the data, which may not be representative. We ran the analysis again with {#articles=400, #timesEach=25}, and this time we restricted the set of allowable articles to those with MCS between 2000 and 3000 bytes. This caused a number of collisions, as shown in the plot below. Yet Profit still reported significant leakage (7.7 out of 8.64 bits) […] The leakage reported by Profit is below what can be achieved. This is due to other distortions that mask the correlation, including 8-padding (explained above) and chunking (explained below). But the reported leakage was strong enough to point us in the right direction. […]

Using active operations to find out the 8 numbers

Eavesdropping on a single benign request cannot reveal the current 8 numbers used by the server. Offline profiling does not help, because the 8 numbers change every time the server is launched. The attacker needs to use active operations for this. One way to do it is to fetch the smallest article enough times. Its size, even with padding, is small enough to guarantee that no chunking will take place, and we know the rest of the fixed overhead from previous experiments. Assuming that the distribution over the 8 possible numbers is uniform, the expected number of attempts needed to see all 8 values is 22 ("coupon collector" problem, n times n-th harmonic number). Also, the probability of 50 attempts without seeing a number is 0.001. In practice, we have seen no case where it took more than 50 attempts to see all values. To stay on the safe side, let us assume that the attacker uses 64 attempts.

We reproduced this as a Profit experiment: Retrieve listAll once, then retrieve the smallest article 64 times, recording the observed SCSs. We used Profit to repeat the whole experiment many times and confirmed that a single TCP session suffices to satisfy all 65 requests, without observing any SYN packets other than the first handshake. The attacker can thus infer the server's current 8 numbers with a cost of 1. (As we will see, we could also use 64 and still be within budget.) For instance, here are the results of two experiments: [..] After subtracting the known smallest article size (1526) and fixed overhead (198), the inferred sets of 8 padding values for the experiments above are {208, 278, 315, 365, 782, 1444, 1913, 2038} and {619, 627, 966, 1550, 1600, 1710, 1892, 2038}, respectively.

Proposed attack

The attacker eavesdrops on the benign user's request (1 listAll + 1 article = typically 1 session, let us assume a worst case of 2 sessions). She can tell where the listAll ends and the article starts thanks to the presence of the client-to-server article request packet. As shown before, she can also use that packet's size to exactly determine the ANL. Using active operations, she can determine the current 8 numbers with a cost of 1. Once the numbers are known, she can infer the (MCS+chunking+fixed overhead) up to 8 candidates. The uncertainty introduced by chunking, as shown above, can be eliminated by considering 9 candidates for each candidate. We thus have 8 x 9 = 72 possible values for the MCS in the worst case.

Using the ANL side channel, she can narrow down the potential articles to the class of articles that have that particular ANL. Then she can further reduce that class to the articles in it that

match one of the 72 possible MCS values. In the absence of size collisions within the class, 72 oracle queries would suffice to guess the right article. But many ANL classes have multiple articles with the same size in them. Note that:

- The ANL class with the largest number of MCS collisions is ANL= 30, in which the most common MCS appears 5 times, the second most common MCS also appears 5 times, and the 3rd most common one appears 2 times.
- For all other ANL classes, the worst-case number of MCS collisions is 3 or less for all of their most common MCSs.

So, in these worst cases, the attacker may need, e.g., up to 5 oracle queries to rule out one candidate, plus another 5 queries to rule out another candidate, plus 2 more to rule out the following candidate, etcetera. In the worst case, this requires (5+5+2+2+2+...+2) oracle queries which is bounded by 150 (i.e., 10+70x2), or (3+3+3+3+...+3) queries, which is bounded by 216 (i.e., 72x3). Even if we assume 65 active operations to extract the 8 numbers, plus 1 passive to eavesdrop on the ANL and SCS, we get a worst case of 281, which is still within budget."

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"[…] two important factors came up during the on-site engagement:

Firstly, we realized that the effects of chunking are much more predictable than we thought. We thought that the effects of chunking and those of padding could interact. In fact, only the actual content is chunked, and the X-Padding is not part of the content, but of the headers, which are not chunked. Therefore, an attacker could (during offline profiling) build a table determining, for each article, exactly how many chunks (if any) will be used when transmitting it, independently of the padding value used. The formula for chunked wire size as a function of actual size described in our original justification remains valid, but this new realization eliminates the uncertainty described in our original document about the number of chunks for an article potentially changing depending on the particular padding value selected. This plays in our favor.

Secondly, and most importantly, Colorado pointed out a fourth source of noise (AES block cipher multiple-of-16-byte padding) that we had not encountered because we used the python requests package, rather than curl, to drive the system, and python requests seems to negotiate

a stream cipher. Since the example scripts use curl, which seems to negotiate a block cipher, we need to take into account the effect of 16-padding, both for the client-to-server request packet (which introduces noise in the side channel into the ANL) and for the server-to-client response (which introduces noise in the side channel into the MCS via SCS).

Our simulations strongly suggest that the attack is still feasible. We did not have enough time to implement the exploit during the live engagement, and due to the absence of this key element, we would need to redo our whole calculations in an essentially different way. Instead, we rely on Iowa's model that shows that in the worst case, there are only very few selections of padding values that could potentially require more than the allowed number of operations, and that even in those rare cases, it is still possible to guarantee 95% probability of success. In brief, Iowa showed that, considering the 16-byte uncertainty in the request, the worst-case bucket for request

size is the 500-byte AES block (16 to 31-char URLs), with about 33K articles. They assumed the worst case of 8 padding values with maximum number of collisions and used a sliding window of size 16 to model the uncertainty introduced in the response. Choosing disjoint windows to maximize the number of unique articles, the 4 worst ones that are within 2048 bytes of each other yielded 41, 41, 35, and 32 oracle queries. In the most extremely unlucky case, this could lead to exceeding the budget. At this point, in order to show that even in those cases the probability of success is 95%, they showed that 40 active queries on the smallest article should suffice to obtain the 8 padding values with 96% probability and 259 oracle queries remaining, with 476 articles in the next-worst-case window of size above 27. They inferred 476/48164 which is 0.988% which may not be reachable with oracle queries. 96% * 0.988% >95%."

Live-Collaborative Evaluation:

Vanderbilt through collaboration observed the predictability in data chunking. Additionally, Vanderbilt noted an effect of the use of python requests versus curl in introducing noise in space side channels.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.8 Poker

A.5.8.2.8.1 Question 004 (SC Space, Intended Not Vulnerable, Take-home: Yes, Live: No)

Take-Home Response:

"We used JAnalyzer to get an initial understanding of the server. The poker server keeps a ThreadLocalRandom instance for each thread (i.e., for each player) to generate the lengths of the padding strings that are added to each message. ThreadLocalRandom is an insecure PRNG. It was designed to reduce contention overhead in multi-threaded environments (w.r.t. a shared Random instance), but other than that, it is just as insecure as Random. Since the algorithm is known, an attacker that obtains the current seed (the most recently returned value) can predict all future values. Consider the message that is sent by the server to the opponent with the empty table. Its content is well-known by the attacker: a base64-encoded rendering of a plain green table with no cards on it, and some static information. The size of the base64-encoded rendering is always 11340 bytes, and the rest of the JSON content of this particular message is always 133 bytes. RECEIVED FROM SERVER: {"opponentsCards":"","padding":"...","holeCards":"","tableCards":"","serverMessageType":"game Status","youAre":"Player 1","rendering":"..."}

The only element with unknown size is the padding. Therefore, if the attacker could eavesdrop on the network traffic and infer the number of bytes received by the opponent, she could infer the padding value by subtraction, and use this number to predict all future padding values. We used Profit to run a client-server interaction multiple times while sniffing the network traffic and to quantify leakage of a wide variety of observables, including aggregated features such as the sum of payloads of packets over intervals between user actions.

We configured Profit so that the secret was the actual value of the padding of the relevant message, plus the known JSON overhead, plus the known empty-table rendering size. Profit reported that the maximum leakage occurred for the following feature: the sum of all payload sizes transmitted in a certain direction (server to client) within a certain phase (corresponding to the message in question). Indeed, as shown in the plot below, there is a strong correlation between this feature and the actual number of bytes that we know (based on known facts) are

being received. The leakage is not perfect, but the margin of error is quite small. Considering 32 attempts with observable sizes between 21,000 and 102,000 bytes, the worst error was 1533 bytes, and the mean error was 573 bytes: […] However, to obtain the exact value of the current seed, the attacker would need to map the side-channel observable to the actual number of bytes received with higher precision—in other words, perfect leakage is needed, or at least within a very small margin of error that can be compensated with <170 oracle queries.

In order to achieve this, specific knowledge of the TLS/SSL overhead is required, which Profit does not have. Taking into consideration all the peculiarities of TLS and the cipher used by the application can help fine-tune the calculation. For instance, we must consider that 80 bytes are lost for every 16KB of application data, that MACs, TLS headers and padding may be added, etc. The specifics are well-documented, and it is important to note that the padding added by TLS in this case is not intended as protection, but rather for practical purposes (e.g., to ensure an integral multiple of the block size), and is therefore deterministic.

Assuming a block size of 16, we believe that an attacker, based on the side-channel information, can infer the number of bytes of the message sent by the server to the opponent to within 16 bytes of the correct number. In the worst case, this could lead to 16 potential calculations that could be disambiguated later on, using oracle queries. Furthermore, let us assume that the attacker, during offline profiling, takes note of the size of the rendered images received by the opponent's client when the opponent's first hole card is served, and when the opponent's second hole card is served. This can be achieved by playing a large number of hands until all combinations are seen (this is related to the coupon collector problem; the expected value is about 22,500 hands dealt until the 52x51 possible combinations are seen), or simply by rigging the server to ensure that all possible card combinations are dealt in quick succession.

Suppose that the attacker builds two mappings, one between each possible single card and the exact size of the rendered image of the table with that card on it, and another between each possible pair of hole cards and the exact size of the corresponding rendered image. The size of a rendered table is correlated with the size of the cards on the table, modulo PNG's lossless compression (when very similar things are on the table, the compression rate improves). The card images are located in the cardface directory, and each has a different size, from 7kB to 90kB. Also, the cards are transparent, and a "skin" is used which can be seen through the transparency. Each of the players is allowed to listSkins (see the available backgrounds) and setSkin (set a background); the latter affects both players. The skins also have different size, with the Cat skin being much heavier, so if using that skin, the size differences between cards can be amplified. Based on these size differences, the attacker would be able to develop the two size signature mappings for the relevant renderings of cards-on-a-table. These mappings can then be used during the actual attack. While we did not have the time to conduct the complete experiment and observe the number of size collisions, we believe that this attack is feasible within the budget due to several reasons:

- The number of combinations of hole cards is relatively small (< 2700).
- The number of possible byte sizes that a rendering can have is much higher (in the range of hundreds of thousands of bytes).
- There are two sources of information that can be combined. If the first-card image yields a unique match, the attack is successful. If there are multiple matches, but the second image yields a unique match, the attack is also successful. And if both images yield multiple matches, their intersection can be considered.

- Even if the intersection still contains multiple matches, the attacker still has oracle queries to disambiguate (modulo oracle queries needed to disambiguate TLS-level padding values on the initial guess of the PokerServer padding value)."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. Vanderbilt noted that with knowledge of the random seed the attacker can predict all future values. The side channel also requires the attacker inferring the size of the padding presumably with knowledge of the game sequence and the restriction in the question that a benign user is playing only a single game of poker with the attacker performing no other actions. Vanderbilt reports a strong correlation between the unknown secret padding, and the known JSON overhead, known empty-table rendering size, and observed encrypted traffic. Vanderbilt claims that this leakage combined with knowledge of TLS/SSL overhead along with a built mapping of card sizes to possible rendered images can be used to leak the secret. The exploit relies on the following:

1. There must for at least 1 of the possible skins available exist this leakage that satisfies the operational budget.
2. The leakage must be present for all possible combinations of TLS/SSL schemes that could exist between the console client and the server.
3. The encrypted padding scheme including block size must be able to be determine by the attacker.
4. For each possible block size (assuming a block cypher is implemented) the should be a sufficient number of Oracle queries to disambiguate potential collisions.
5. Finally, all this must be able to be used to determine the exact value of the current seed.

We believe that learning the opponent client's PRNG seed requires passive observations of the client's output over many games. The challenge question restricts the opponent to only playing a single game, ruling out this attack possibility.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Two Six Labs showed us full results confirming what we had already found out through our simulations—that, if it weren't for the padding, the fingerprinting attack on the image sizes would certainly be feasible. The question is, are we sure that we can defeat the padding? We initially thought so, but we then realized that we aren't.

Colorado and Iowa pointed out that inferring one instance of the opponent's padding value from the known rendering size does not suffice to know the full state of the PRNG, since the padding is a pseudorandom integer between 5000 and 95000 (hence truncated w.r.t. the full state of the generator). Indeed, we would need many instances of padding values to crack the PRNG and predict the sequence.

GrammaTech proposed using multiple successive padding values inferred from the attacker's own (thus, known) rendering sizes to crack the PRNG. However, we pointed out that each player is served by a different thread, and that a separate instance of ThreadLocalRandom is returned when each thread calls ThreadLocalRandom.current(), thus leading to separate PRNG states. Information about the attacker's padding values should not be helpful.

GrammaTech also proposed playing multiple hands, then reconstructing everything that happened from the a-posteriori-known results of the hands, and then inferring multiple padding values in the opponent's previous hands in order to predict the sequence for the next hand. However, we pointed out that the question statement only guarantees that a single hand of poker will be played. There is no guarantee that multiple hands are played, let alone that one or more hands will be played before the one that we want to attack.

After a long discussion, all teams reached general consensus that the fingerprinting attack is feasible if we can defeat the padding; that, due to the magnitude and uniformity of the padding there is no hope of reaching the desired probability of success without defeating it; and that, so far, no one has found a feasible way to predict the random sequence from the side channels that we have found. As a consequence, we are changing our answer to NO."

Live-Collaborative Evaluation:

Following collaboration, Vanderbilt correctly observed that the padding coupled with the question restrictions on a single observation prevent a potential SC Space vulnerability. Vanderbilt correctly concluded that there was not an intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.8.2 Question 031 (SC Time, Intended Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"We used JAnalyzer to get an initial understanding of the problem. The poker server attaches a rendering attribute to every message that includes an encoded image for drawing the table on the client side. To create this image, the server reads the images of Hole Cards from the cardface directory and put them in the right position on the table. The images in cardface directory have different sizes, from 7kB to 90kB. We suspected that the processing time of rendering depends on the size of cards' images. Therefore, using JAnalyzer, we identified the methods drawTable() and base64EncodeImage() located in Renderer.java that upload and process the cards' images. Then, we instrumented the server side to measure the time before and after performing these two methods.

We implemented two experiments: (1) exp1 sets the first hole card to 2 of Spades, with size 7kB, and (2) exp2 sets the first hole card to King of Heart, with size 90kB. These two experiments have the highest size difference and thus their time difference should be the highest. Also, since different card skins have different sizes, with Cat skin being the much heavier than others, we used the Cat skin to amplify the processing time and the delay. You can find the experiments in the attachment.

However, running the two experiments multiple times, we observed only small differences in the time, about 20 microseconds. Based on these observations and having 50 other image cards with sizes being between these two cards, we conclude that it is highly impossible to detect a strong difference between different cards on the server side as well as on the network."

Take-Home Evaluation:

Vanderbilt focused their analysis on discerning the difference between individual cards and missed the timing difference in how cards are retrieved (new render vs retrieved from cache).

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Incorrect

A.5.8.2.9 Searchable Blog

A.5.8.2.9.1 Question 027 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Using JAnalyzer on this application quickly led to identifying locations with potential algorithmic complexity. The main entry point is BlogSearchController.submitUserBlog which performs several trivial actions, followed by a call to RankingService.performRanking This eventually calls MatrixRoutines.estimateStationaryVector which attempts to approximate a stochastic vector of the adjacency matrix of all blogs currently in the system using a seed vector (either a fixed one or one generated randomly). The loop that estimates this vector has an unstable exit condition: it iterates until the distance between v and (M * v) is below a hard-coded epsilon.

An exact stochastic vector would yield v = (M * v). To use more common language, the eigenvalue of v is 1. The algorithm used to calculate this is known to converge slowly for some combinations of M and v. The adjacency matrix is calculated by analyzing the blogs in the system and identifying links within the blogs. It seems that if the adjacency matrix could be manipulated by adding another user blog such that the estimation of the stochastic vector does not converge below the epsilon value within the budget, the vulnerability would be triggered.

While the combination of a statically assigned initial search vector, a client-controlled transformation matrix, an unstable exit condition, and known degenerate cases in the computation algorithm, we were not able to confirm the existence of the vulnerability. We were, however, able to use our tools to quickly identify a likely source of a vulnerability and the general conditions required to trigger said vulnerability. Based on this information, we can state that this challenge problem contains a vulnerability that could be exploited by a hostile actor."

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"We collaborated with the Colorado team. Although we answered YES originally, we didn't come up with an effective exploit. After collaboration, we understand the program more, and can generate an exploit. We keep our original YES but update our justification. We first modify the program to remove the Spring Framework used in the program so that only the application code remained and replaced the interface with a simple command line interface. This allowed the user to perform the same user interactions as with Spring Framework, and it can also let us use our Janalyzer and Kelinci tools.

Janalyzer (as well as visual inspection) pointed out the com.bbn.MatrixRoutines.estimateStationaryVector() method has a suspicious loop. The com.bbn.RankingService.performRanking() method initially sets up the adjacency matrix A from the list of provided blogs. There are 729 blogs in the data folder plus the userblog.html that is copied over initially as an empty file, thus a total of 730 entries. This makes matrix A a 730 x 730 matrix initially. When a new blog is added, it copies the new file to the /blog/userblog.html file, calculates a new adjacency vector for it and replaces the last column of the matrix with this entry, so the matrix always remains at the same size with only the final column of data updated for each blog entry added. This allowed us to add another user command to specify a 730-entry vector value that is copied to the final column of the adjacency matrix prior to a call to estimateStationaryVector(). […]"

Live-Collaborative Evaluation:

Vanderbilt strengthened their response from collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.10 SimpleVote 1

A.5.8.2.10.1 Question 019 (AC Time, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"There are 3 threads in the server: //Main// that runs the initialization actions, //StacMain$1// that handles the shutdown hook (idle until the server is exited or aborted, and //CompilationChore// that handles the network requests and responses. This last one is the only thread that effectively runs all of the user interactions with the server, which would allow any user request to hold up another request. Therefore, if we can find a user command that simply takes a long time to respond to the attacker, a benign request from any other user would also be deferred. […]

First, we look at the recursions list for issues. //TST.collect// is only called from LZW.decompress, which is never called, so we can eliminate that. Next are the methods //TST.get// and //TST.put// are called from the following:

- AdminGuide.handlePost

- ConfirmationGuide.handlePost

- CompilationChore.run - scheduled to run once every 60 minutes to update all surveys

These also do not seem to have any looping or recursion issues. The next case is DirectVoteLoader.loadVoters , loadSurveys , and loadBallots . These are called only during startup so are not really able to be exploited by an attacker. They are used to initialize the database with the entries from the voters.json file and the elections and ballots directories in the content folder of the data directory passed in the "-d" option when starting the server. These cannot be a source of a vulnerability unless the content of the data folder is modified by the attacker, which is out of scope. Next, the methods basicvote. PermissionAuthenticator.assessRange and PermissionAuthenticator.confirmEssence are called when a registrationkey message is received from the network. There is a 12-character String associated with the field "registrationKey". The characters should be all numeric with the additional chars A , C , and E, which get transformed into the values 7 , 6 , and 3 correspondingly by the method confirm. If the lower 6 digits are < 995, it calls confirmEssence

passing as arguments a String of the 1st 6 digits, the integer value of the lower 6 digits of the string (this will be a positive value 0 - 995), the negated integer value of the lower 6 digits, and boolean set to false. isVotingActivated is called to determine if assessRange will be called. This method uses a synchronization lock to access a flag, and may wait for up to 5 sec if the resource is currently locked. However, the actual time it will wait should be negligible since the competing action that can take the lock simply sets the votingActivated flag, releases the lock and issues a notifyAll which will terminate any other thread that is waiting. isVotingActivated always returns true, so it will always call assessRange. assessRange is then called from confirmEssence and is passed the same argument values with the exception of the last. This boolean value will be be set based on the boolean value received and whether we get a "match" of the upper 6 digit String value to some manipulation run on the integer represented by the negated lower digits. The "matching" process consists of converting the negated integer value to a floating point double value and passing it to FunkyFunction.funkyFunction. The returned double is then passed to matches along with the upper digits String. matches uses BigInteger math to convert the floating point value with 6 digits of precision to a BigInteger value, then convert that into an array of characters. The array of characters is then compared to the content of the string passed. A lot of busy work for a simple string comparison.

The method assessRange then compares the 2 numeric values passed and if the lower digit value is > the negated version, it calls confirmEssence again with the same parameter values except the negated integer value is incremented by 1. The last argument value, the boolean, has no effect on the recursion, but it is possible that the funkyFunction may be very time consuming. The recursion of confirmEssence will then recurse until the 2-integer values match, which will be when the 1st integer value is equal to 995. This will cause a recursion of 1990 times. Since we are looking for a time consumption of 300 sec, this would require an average of 150 msec of processing time per loop. The method funkyFunction calls Table.obtainEssence many times, which will perform an Exception.getStackTrace for all cases where the integer value < 0 (995 occurrances) and uses BigDecimal math functions, both of which can be time consuming. This may be a possibility. The registrationkey.json contains the registration key values used in the election. There are only 2 entries in the listing that have the lower 6 digits <= 995 and those are:

electionId: 400, index 19 has a registrationkey value of "0321A3 000906"

electionId: 491, index 3 has a registrationkey value of "185744 000905"

The first of these entries also happens to be the registration key assigned to user KathyRBown@rhyta.com for accessing the Baltimore City Democratic Primary, which does run without any appreciable delay, so this is undoubtedly not a vulnerability. We crafted some registration keys and timed how long it took for the server to validate the key. First the table needs to be increased, so we send two keys where N=995 and N=1002; this makes MAX_N increase to 1009. Then we send a key where N=1007, which takes the server 412 seconds to validate. […]”

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.10.2 Question 025 (SC Time/Space, Unintended Not Vuln., Take-home: No, Live: No)

Take-Home Response:

“From the available actions a voter has, the only way to access their ballot for a particular election is to log in with both their username and password (through the /login webpage), view an open election from their homepage, then provide a valid registration key for that election. Crucially, the validity of the registration key is not tied to the voter it belongs to. If an unsuspecting voter somehow leaked their registration key to an attacker, the attacker could then access and modify the client victim's ballot. Thus, the secret in question is the registration key. From the question description, we know that the target voter signs into the application once and views the target ballot at least once during their session. In terms of interaction with the application, the client logs in through the /login page, is redirected to the /elections page (summary of current/past/future elections) and clicks on one of the current election, leading to /registrationkey/#electionid, where #electionid is the unique id corresponding to the election in question. The target voter then supplies their registration key, which, assuming a valid registration key, allows the target voter to edit/finalize their ballot. With this knowledge we used JAnalyzer to track down the sequence of functions corresponding to the above actions, focusing on those related to the registration key: specifically, PermissionAssessGuide.handlePost() corresponds to the /registrationkey page which sends the submitted registration key to PermissionAuthenticator.confirm() for validation. We next discuss the structure and validation of a registration key.

A valid registration key consists of exactly 12 alphanumeric characters which must conform to a non-trivial pattern based on position and value. Essentially, two different strings kStr , nStr are encoded within the key, each being 6 characters long. For a registration to be valid, both kStr and nStr need to be valid in both their structure and relation to each other. Specifically, nStr will be casted into an integer n and must be within the range [0,MAX_N), where MAX_N is initially set to 995 but can grow over the course of the applications lifetime; n must also be divisible by 7. The relationship between kStr and n is validated within the recursive structure of the methods PermissionAuthenticator.confirmEssence() and PermissionAuthenticator.assessRange() : […] Specifically, confirmEssence() does work through FunkyFunction.funkyFunction() , then calls assessRange(); assessRange() increments a counter a (initially set to - n) and calls confirmEssence() if a < n or simply returns if a >= n. It follows that the number of times this recursive cycle continues is directly proportional to 2 n . Note that a registration key is only valid if the last recursive call returns true. For each value a in [-n,n], funkyFunction() returns a value V_a which is computed based on the recurrence relation V_i+1 = 4*V_i*(1-V_i), where V_0 = 0.375 and i ranges from [0,a]. Crucially, funkyFunction() uses an internal data structure to store the values of V_a, so rather than calculating the recurrence relation from 0, it instead uses the closest value to a that it has computed thus far as its initial value. From the recursive cycle and the properties of funkyFunction(), we can deduce that each value of n corresponds to a unique kStr . Additionally, the number of recursive calls increases as n increases. This increases the time it takes to validate a registration key, leading to a possible side channel. We used the packet analysis portion of our tool Profit to analyze the time it takes to validate a registration key as n

increases: see the attached file simplevote_1_timing_graph.png (Y-Axis shows time in seconds, X-axis shows the value of n divided by 7. From the graph we can clearly see a timing leakage, possibly allowing an attacker to reconstruct a target victim's registration key.

While there may be leakage in the value of n , and n may produce a unique kStr , it is possible for multiple registration keys to map to the same values of n and kStr . A registration key belonging to a user may contain both letters and numbers, while the values of n and kStr must be digits. In the initial validation phase, the characters corresponding to those of kStr can be any digit 0-9 or the letters A,C, or E. As kStr is built, A is replaced by 7, C is replaced by 6, and E is replaced by 3; e.g., if the initial characters of the registration key corresponding to kStr are "55422A" then kStr will be "554227." This means that multiple valid registration keys will map to the same values of n and kStr. In the worst case, it is possible that all 6 characters of kStr could have initially been the characters A,C, or E, producing 2^6=64 registration keys, with only one of them being valid for the particular election in question. There seems to be no feasible way for an attacker to determine which of the 64 keys, aside from the use of oracle queries: in this case, as the operational budget is 25 operations, an attacker can try 25 out of 64 possible registration keys, giving a ~39% probability of success, rendering any attack infeasible (due to the required 70% probability of success)."

Take-Home Evaluation:

The challenge program contained an intended vulnerability whereby a registration key is transformed to an integer in a deterministic manner, and the runtime for processing a registration key is linearly correlated to the value of the key. The side channel strength is dependent on the secret range, and the application code must unambiguously provide this. We intended for there to be a 1-to-1 mapping between a registration key and a translated integer value of the key. However, in the application code 3 specific characters in the registration key are replaced with 3 specific numbers during the transformation from registration key input to an integer. The challenge program therefore allows for 2^6 possible registration keys to map to the same integer. There are not enough oracle queries to resolve this. This unintended not-vulnerable case was identified by Vanderbilt.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.11 SimpleVote 2

A.5.8.2.11.1 Question 005 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We used JAnalyzer to identify the entry and exit points in the server-to-server communication. We then traced the secret (in this case query responses) towards the exit points. The application uses its own implementation of the longest-prefix-based compressor before transferring the packets between the two servers. The longest-prefix-based compressor is based on the generation

of a "formation" that builds a tree of prefixes from the non-multiple-choice questions. The non-multiple-choice questions are not related to the secret in this question. Then, the multiple-choice query data with responses and the summary across all currently finalized ballots in a given precinct are used to build a display message. This display message is then compressed using the above-mentioned formation tree. The result is sent to the other server, which on the arrival of the packet copies the same formation and uses it for compressing its response -- where the response being a result of combining the reports of both servers -- and send the response back to the first server. The attacker can only observe the size of the generated packets exchanged between the servers in this connection.

We performed dependency analysis to trace whether it is possible to separate out any particular candidate and his/her corresponding votes within the display message or consequently within the compressed output. There is no such dependency observed in the implementation. Also, the compressed function will almost never get compressed significantly when the update on ballots is observed between the servers when they exchange current summaries. In fact, to make the compression significantly visible the current string representing the complete display message with the reports in it should match an existing prefix and if we expect the numbers of votes changing – no significant matching of the prefix is observable. Note, the prefix is matched within the entire display message string, not in consecutive chunks of it. Also note, that the number of votes is padded to a fixed size when generating the display message. In addition, considering that the number of candidates remains unchanged (it is less than 20) it made us suspect that there will be a very little change observable in the packet sizes that would leak information about specific candidates and their corresponding votes.

We ran experiments on the reference platform with various ballot sets and updates and used Profit to observe the traffic for further analysis. As expected, in our experiments we did not observe any leakage."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attacker can craft inputs to influence the compression scheme in the synchronization messages. This can be used to perform a binary search over the secret space. Vanderbilt flagged the compression scheme as potentially vulnerable but analyzed the application looking for a side channel that exists in the set of tested cases missing that one can be triggered. This input was not amongst the tested set of inputs.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We collaborated with the team from Iowa who demonstrated their exploit to us. They noted a few key points:

1) In the election tallies that the servers share periodically, a candidate is only included if the percentage of votes they received is over a certain INCLUSION THRESHOLD.
2) While this threshold defaults to 0, it can be changed and controlled by an attacker (assuming no one else is voting during the attack). This can be done through the attacker voting in an election that has a free response form. If the attacker's response begins with a space, then the INCLUSION THRESHOLD is controlled by the number of spaces in the response. We manually confirmed this through code inspection of

ReportMessageCreator and EditRecordManager and ran a few examples on the simple vote server that demonstrated this.

3) As we noted in our original response, packets are compressed before they are sent between the servers and this compression uses a tree built of the responses to the non-multiple-choice questions. This means that if we use a candidate's name as an answer to a free-form question, we can tell if that candidate also appears in the report i.e. if their percentage is above the INCLUSION THRESHOLD in that precinct.

From these observations, an attack can be constructed. First, we login and determine the size of the reports from both servers without any modification. This can be done either by waiting an hour or by forcing the server to update by submitting a ballot ten times. Then we perform a binary search as follows. We enter as an answer to a free-form question the name of the target candidate preceded by a space and the required number of spaces to set the inclusion threshold as desired. We then wait for an update and observe the size of the response of both servers. Based on the difference in size between what we observe and what we expect given our baseline, we can determine if the target candidate is included in the reports. We then update the inclusion threshold as appropriate and continue to perform the attack, eventually using oracle queries when it becomes cheaper to do so to reach the final result. […]"

Live-Collaborative Evaluation:

Through their collaboration with Iowa, Vanderbilt identified the intended vulnerability. In this response, Vanderbilt noted that the precinct results are periodically shared between precinct servers. While this is true, an attacker can also trigger this synchronization. Both avenues of attack are valid.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.11.2 Question 016 (SC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

"Given the actions available to voters, the only way to access a voter's profile is to login with both the voter's username and password (through the /login webpage), then register a GET request for the /profile webpage. This means an attacker only needs the username & password of the target voter (victim). Since the username is known, it follows that the password is the secret value in question. Using JAnalyzer to extract the CFG and the knowledge that the secret value is the password, we identified the entry point of the secret value to be the LoginManager.handlePost() method, which is called when the target voter enters their username and password to the application through the /login webpage. The supplied password is checked against the stored password within the method body in the call to TokenPasswordChecker.processPassword(). The password checking functionality turned out to be more than a simple for loop that checks for matching characters. Rather, the process is a nontrivial one where not only is the given password checked against the stored password, but also an intermediary string token is created during the process. If the two passwords match, token is discarded and the client voter is allowed access to their profile and elections; otherwise, the client is denied entry and token is sent back to the client along with an error message saying the password and/or username was incorrect. It is important to note that the password must be between 5-15 characters in length (inclusive). We hypothesized that if there was any leakage of the secret value, it must be through the returned token. Our investigation into the possible values

of token revealed not only the structure of the string, but also how it relates to the secret value. token is constructed to be a sequence of spaces (somewhat random), concatenated with the stored/correct password, concatenated with a sequence of spaces (same number as before), concatenated with the given password. In other words, if the stored password is "abcde" and the password to be checked is "12345" then one possible value of token is: " abcde 12345", where the number of spaces before each password is 15. (It is always the case that the number of spaces before each password is equal). As mentioned the number of spaces is random, however the random is seeded by the correct password, which means given a username the size of the space padding in the corresponding token is fixed. After creation, token is zipped/compressed using the java.util.zip.Deflater library, which is essentially equivalent to using gzip. As mentioned above, if the given password is incorrect, then token is sent back to the client. Crucially, this value corresponds to the compressed value of token, but only after each byte of token is randomly modified.

Given the structure of token and that the size of the compressed token is preserved through the random byte transformation, the application is vulnerable to a prefix attack, such as CRIME [1]. In our prior work [2], we investigated such side channels and proposed a method for automatically synthesizing adaptive attacks to exploit applications similar to CRIME. We apply such an analysis here and confirm that there is an attack within the operational budget. Additionally, we used the packet analysis portion of our Profit tool to experimentally verify the leakage is observable. The size of the returned token is indeed reflected in the size of the corresponding packet (through which it is returned). In fact, this token is also observable in the url of the corresponding error page. A valid password contains between 5-15 (inclusive) characters, where each character must be in the ASCII range of 33-126 (inclusive). In the worst case, the longest password is 15 characters, with each character being one of 94 possible choices. Since the compression is performed by matching the prefix we start guessing the password character by character (spaces match already) and the most compressed output will correspond to the match at every iteration. A prefix attack that determines each character of the password starting with the first character can uncover a password of length 15 in 94*15 = 1410 operations, with one operation per network observation plus one oracle query = 1411 operations, which is within the budget of 1500 operations."

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.12 STAC SQL

A.5.8.2.12.1 Question 013 (AC Space, Intended Vulnerable, Take-home: Yes, Live: Yes)

Take-Home Response:

“JAnalyzer's news functionality reports allocations in SQLParser.parseTokens and SQLBaseAction.parseCommaSeparatedValues . Notably, the latter of these two functions allocates a new ArrayList, tokenizes the input on any commas, and adds the resulting literals into the ArrayList. As such, it stands as a candidate for an algorithmic complexity in space. Several other allocations listed in JAnalyzer are in the various subclasses of SQLAction , which are reached when parseTokens iterates through all the known actions. With this knowledge in mind, we were able to focus our search to these subclasses. Most of the problem is the combination of stuck values and writing logs. Attack in a nutshell: In SQLInsertAction class, there are two fields that can be stuck with old values (although other action classes also have this kind of fields, we choose insert action because it has two fields that can be exploited). We let List<String> columns and List<String> values have a bunch of nonsense strings, and repeat some valid insert operation such as the shortest insert help. Each insert help will render "INSERT " + this.tableName + " " + this.columns + " " + this.values to be written into the log file logfile.log. (Note that if we want to speed up things, we can also let String tableName stick to a long table name.) […]

Repeat insert help in a string -- EACH insert help ends with a new line (WHY? because we want to amortize the overhead in as many as possible insert help's). If the optional step is used, about 532 insert help will be used (about 6.2KB + 2.9KB + several MAC frame's overhead = 9.1KB + overhead in total) to reach the limit. If the optional step is not used, about 710 insert help will be used (about 8.3KB + 1.9KB + several MAC frame's overhead = 10.2KB + overhead in total) to reach the limit.”

Take-Home Evaluation:

The challenge contains an intended vulnerability. Vanderbilt appears to have discovered an unintended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.13 Stegosaurus

A.5.8.2.13.1 Question 003 (SC Time, Unintended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“The main method responsible for hiding the message inside an image contains a routine that has time side channel vulnerability in it. We located the "hide" method by first identifying the exit points and following backwards towards the input plaintext secret using JAnalyzer. Before executing the "hide" routine, the plaintext gets fuzzed by adding a slight (randomized) noise to the plaintext. The noise depends on the plaintext, the key provided by the user and the size of the image, and it does not change the length of the plaintext. The runtime of the noise adder procedure is quadratic in the length of the message (uses bubblesort), but otherwise is independent from the characters in the plaintext. The "hide" routine is performed for each bit of

the fuzzed plaintext message and the execution time of the routine depends on whether the fuzzed plaintext bit is 1 or 0, with taking significantly longer to process bit 1.[…]

We apply modular analysis to the hide method, which returns equivalence classes of the input values the method operates on. The variable "value" is one of such input variables, that depends on the secret message (this.text) and equals to the value of the current bit of the message. Thus it is either 1 or 0. Later the routine computes "value *= 3;" and increments the variable "pkCopy" by "value^80", where "pkCopy" is the 128-bit key provided by the targeted user along with the fuzzed plaintext. Thus if "value" = 0, "pkCopy" does not change; while when "value" = 1, "pkCopy" becomes considerably larger. This is crucial for the execution time of the "for" loop that follows:

```
for (pkCopy = pkCopy.add(BigInteger.valueOf(offset));
pkCopy.compareTo(BigInteger.valueOf((long)maxOffset)) > 0; pkCopy =
pkCopy.subtract(BigInteger.valueOf((long)maxOffset).multiply(this.perf))) {
;
}
```

Where "perf" is a hardcoded large number 19999999210123456789012345 67. The loop is basically a slow modular computation of "pkCopy mod maxOffset" -- the loop reduces the value of "pkCopy" with the steps of "maxOffset*perf". Here, "maxOffset" = 3* width * height of the image where the message is to be hidden. When the value of "pkCopy" is incremented by "3^80", this "for" loop takes significantly longer time to execute versus the case when the value of "pkCopy" is only the 128-bit key. Thus, the hide function leaks the number of 1's in the processed text. The size of the plaintext is 512 characters and no restrictions are imposed on accepted character set. The application is independent from the actual image, it modifies values of the pixels based on the offset set, generation of which depends on the key provided by the user and the size of the image used. Extracting the width and height of the image by observing only the timings is not feasible given the accepted image format is jpg and thus allows compression. The code also has no other way of extracting neither the size of the image nor the key provided by the user. However, we found that the progress reports used in the application can be intercepted by the attacker, as all that is needed to load the progress reports is the "ThreadID" of the particular user. ThreadIDs are distributed in the order of the lowest number available. Since the question assumes that the victim user and the attacker are the only users communicating with the server during the attack, the attacker knows that the victim’s ThreadID is 0. As a side note, we also noticed that the application allows existence of "zombie" processes -- which are the processes that were interrupted while performing an operation on the server. Since the victim and the attacker are the only processes communicating with the server we also conclude that there are no zombie processes during the attack. Thus, the attacker knows the "ThreadID" of the victim is 0. We modified the shell script provided for the "hide" operation. The attacker keeps requesting the status updates for victim's ThreadID process, as follows […]

The progress update reports return current bit of the fuzzed plaintext processed in the moment when the request for the progress update was made, along with the total number of bits in the plaintext. The poll requests do not have side-channel in time. Thus, the attacker can obtain the progress updates in small chunks of fuzzed plaintext bits. The number of bits in these chunks depends on how many 1-s were processed in this chunk. Given that the amount of time spent on

processing each 1 or each 0 is of the same magnitude, the attacker can compute with high accuracy the number of 1-s and, thus, the number of 0-s in each chunk. The idea would be to bruteforce these chunks of the fuzzed plaintext (and bruteforce the fuzzing). The worst cases to brute force are those with equal (or nearly equal) numbers of 1-s and 0-s in the chunk. E.g. in strings comprised of the characters "t", "q", "x". Each of these characters contain four 1-s and four 0-s. We performed experiments on the reference platform, running the attacker's script to intercept victim's "hide" requests for plaintext strings containing "t", "q", "x" characters. The intercepted progress reports returned chunks of about 45-50 bits. Such scenarios for bruteforcing the positions of about 22 occurrences of 1-s is the binomial coefficient C(45,22) (of the multitude of 10^13) -- which is way beyond the 5000 budget. To conclude, if the platform setup would allow to get more refined progress outputs, fitting in given budget would have been feasible."

Take-Home Evaluation:

Vanderbilt identified that the challenge program contains a leak in encoding the hidden message: that the runtime for each bit can be used to distinguish a 1 bit from a 0 bit. Additionally, Vanderbilt observed that the attacker can query the heartbeat thread (using question constraints to determine the ID). Rather than using this information to leak each bit individually, Vanderbilt flagged a weaker side channel that can be used to leak the number of 1's and the number of 0's. The analysis is accurate until the claim that the heartbeat (poll) requests do not have a side channel in time. This leads down the path of reconstructing the message from the number of 1's and 0's in the chunks of plaintext bits retrieved via the heartbeat (poll) requests. This appears to be down to Vanderbilt missing the existence of the thread interaction that allows the attacker to perform a segmented extraction of each bit individually. However, it was discovered that the intended vulnerability is of insufficient strength.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.14 StuffTracker

A.5.8.2.14.1 Question 030 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We loaded the program in JAnalyzer and looked at the allocations that occur inside nested loops. There are hundreds of such allocations at the 5, 4, and 3 levels. We noticed that the majority of these happen in the serialization and de-serialization routines. We identified several locations where the allocation of an array happens based on user input that is provided indirectly. For instance, the com.ainfosec.StuffTracker.impl.deserializeArrayIntB method (below) allocates an integer array using a size based on user data that was previously serialized:

[…] This led us to investigate how we could trigger methods like these. However, when we inspected the serialization routines, they appear to serialize only statically specified types,

namely Strings, and thus we were not able to trigger the serialization of the types that looked more malicious, such as nested arrays.

Note: to load the program in JAnalyzer, use the "entry" file in the "assets" directory in the "Additional Entries" field of the JAnalyzer load screen. Also used the "APIsUsed.txt" file in the "Used APIs" field. The "Libraries" field should contain rt.jar (the jar containing the Java class libraries), and the "Application" field should contain the path to the StuffTracker-1.0.jar. A screenshot of example JAnalyzer settings is provided in the assets directory."

Take-Home Evaluation:

Vanderbilt missed the intended vulnerability in the SDL parsing algorithm and resulting compression/decompression algorithm.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We collaborated with the Colorado team, and they convinced us that there is a AC space vulnerability. We will change our answer from NO to YES. Basically, we can use the strategy similar to lol bomb. The problem is in the com.ainfosec.StuffTracker.parse.SDLParser.parseStuff() method, which directly uses the SAXParser to parse the string "xmlin". It's not a "validating" parser, so we are able to stick in nested elements anywhere."

Live-Collaborative Evaluation:

Vanderbilt worked with Colorado and identified the same out-of-scope billion lols-style vulnerability as other teams.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.15 Tawa FS

A.5.8.2.15.1 Question 012 (AC Time, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"We used JAnalyzer to get an initial understanding of the problem. There is an algorithmic complexity vulnerability in time in the defragmentation operation supported by the Tawa_fs server. We confirmed this vulnerability by an experiment where the attacker submits two large defragmentation tasks from two different sessions, which cause denial of service for a benign user who cannot login to the system. Here is the sequence of operations by the attacker:

Attacker commands:

1. Registers
2. Logins twice using his credentials, i.e., has two open sessions.
3. Uploads N empty files. For now with N=1000 the attack works.
4. Randomly deletes twelve of these empty files from the filesystem.
5. Uploads three big files that each goes over 4 inodes in the fs. (12,300 Bytes).
6. Deletes a few more empty files.
7. And, finally, calls defragmentation simultaneously twice form its two sessions.

This sequence of deleting some files, then adding large files and finally deleting more files forces the file system to be fragmented and thus the defragmentation being triggered. By deleting 12

empty files randomly from the filesystem, some unused inodes (holes) are generated in the filesystem. Then, when adding three big files, they are going to be fragmented and put in the holes. The defragmentation method is only triggered when a file goes over at least 4 inodes and the position of its inodes in the filesystem are not sequential. Finally, by deleting a few more files, the attacker again generates some holes in the filesystem. Trying to be in the budget, we chose to add three big files with size of 12300 Bytes that go over 3 inodes (inode size = 4096). The defragmentation for this setup with N=1000 takes about 3 mins and the benign user cannot login during this time. In the attachment, you can find the shell script for implementing this attack.

Use of KelinciWCA

We ran KelinciWCA to find configurations that trigger worst-case in defragmentation. KelinciWCA extends Kelinci with prioritization of costly paths. Costs are collected on the Java side, then sent to an extended version of AFL. The fuzzer maintains a current highscore with respect to the used cost model. When creating the next generation of inputs, the fuzzer selects ancestors from the set of inputs from the previous generation that either lead to the execution of previously unexplored program branches or to a new highscore. The chance of an input being selected from this set depends on its score, as recorded on the Java side. We used a new functionality of KelinciWCA, user-defined costs. For that we (manually) instrumented the defragmentation method with calls to the special Kelinci.addCost(int) method. This functionality allows a relationship between input and cost about which a machine can reason. In addition, it makes it possible to define cost for specific methods and classes and not the whole application. We implemented a very general driver that reads a random sequence of operations (create, delete and grow) and file sizes from an input file, such that: The first byte of input file indicates the operation, Create, Delete or Update/Grow.

If the operation is Create then:

a) reads the next 4 bytes of the input file to get the file size
b) creates a dummy file with that size, adds it to the filesystem
c) then it reads the next Byte from the input file to get the next operation

If the operation is Grow then:

a) reads the next 4 bytes of the input file to get the file size, s,
b) finds the smallest file in the filesystem, ss, check if (s>ss),
c) if bigger then creates the file with the same name of the smallest file
d) if smaller then just creates a new file with size ss
e) then reads the next Byte from the input file to get the next operation

If the operation is Delete then:

a) it deletes the smallest file in the filesystem,
b) then it reads the next Byte from the input file to get the next operation

You can find the driver in the attachment. To run KelinciWCA we generated an input file including the worse case that is described above. However, after running it for 13 hours, it could not find any configuration that its cost exceeds the cost of that worse case. We believe the reason is that the driver is very general and it has many parameters. For the future work, we will implement several experiments and fix the sequences of operations or limit the random operations.

Budget

To trigger the loops in the fragmentation method, we should have some big files that go fit 4 inodes in the fs, i.e., the file size should be 3*inode_size + delta, where inode_size=4096. In our experiments we used Big_file_size=12,300Byte. Considering the average HTTP packet sizes for a request captured by Profit (~ 650Bytes), then sum of the PDU sizes of the application requests sent from the attacker to the server would be 1000*650Bytes + 3*12,300Bytes= 686.9 kB, which is above the budget, 40 kB."

Take-Home Evaluation:

The challenge contains an intended vulnerability the defragmentation process. An attacker can cause self-recursion using up the JVM stack resulting in an AC Time vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Similar to us, Iowa State University, and University of Colorado had identified this vulnerability that only two threads are allocated to the application and also that there is an algorithmic complexity vulnerability in time in the FileSystem.defragment(). However, we had not identified the optimal sequence of operations that can exploit this vulnerability to be in the budget. University of Colorado presented an exploit which uses the vulnerability in FileSystem.findAllFile(path) when the path is a nested directory and all folders have the same name, e.g., "/a/a/a/". Then, calling the defragmentation in this nested directory, causes an infinite loop in FileSystem.findAllFile() that keeps searching for files in this nested directory. As we reported earlier, the reason that KelinciWCA could not find the worst case is that there are many inputs (operations and file sizes) that for the fuzzer is hard to find the worst cases. This gives us some insights to improve the fuzzer to work on applications with several inputs."

Live-Collaborative Evaluation:

Vanderbilt noted in their take home response that they were hinting at the intended vulnerability but did not have a way to exploit it and therefore responded "No". Collaborating with Iowa and Colorado they were able to craft and exploit within the input budget.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.8.2.15.2 Question 020 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"Attack 1

The file system creates a user.fs file for a user with a fixed size 100MB. We found that all operations including creation, sharing and even deletion allocate space on the disk. However, the size of this file is only checked before uploading a file in order to avoid uploading a file when the file system is full. Based on this, an attack is possible where the attacker creates many empty files and folders, shares them and then deletes them. All these operations allocate some space on the disk while the application does not check the size of filesystem before performing these operations. Therefore, they will not be prevented even if the filesystem size increases above 100MB.

Attack 2

We also found a bug in implementation of fs.getSpaceLeft() method. We noticed that when creating an empty file, two inodes are allocated to it which is equal to 2*4096 Bytes. However, the computation done in fs.getSpaceLeft() shows that only 511 Bytes is consumed. This means that the attacker can upload files with size up to 16 times (2*4096/511) of the fs size (100MB). This is equal to 1.64GB which is above the 1.5 GB budget. We implemented this attack and it is attached. However, creating empty files is very lengthy and even though the experiment is run for three days, it is not done and we could not validate this attack.

Use of KelinciWCA

We ran KelinciWCA to find configurations that trigger worst-case in space. We used the new functionality of KelinciWCA, user-defined costs to define a cost in the write() and writeByte() methods in BlockDevice.java. For that we (manually) instrumented the defragmentation method with calls to the special Kelinci.addCost(int) method. We implemented a very general driver that reads a random sequence of operations (create, delete and grow) and file sizes from an input file, such that: The first byte of input file indicates the operation, Create, Delete or Update/Grow.

If the operation is Create then:

a) reads the next 4 bytes of the input file to get the file size
b) creates a dummy file with that size, adds it to the filesystem
c) then it reads the next Byte from the input file to get the next operation

If the operation is Grow then:

a) reads the next 4 bytes of the input file to get the file size, s,
b) finds the smallest file in the filesystem, ss, check if (s>ss),
c) if bigger then creates the file with the same name of the smallest file
d) if smaller then just creates a new file with size ss
e) then reads the next Byte from the input file to get the next operation

If the operation is Delete then:

a) it deletes the smallest file in the filesystem,
b) then it reads the next Byte from the input file to get the next operation

You can find the driver in the attachment. We ran KelinciWCA so the fuzzer can find the worst case where the cost in space in maximal. However, after running it for a day, it could not find the worst case configuration. We believe the reason is that the driver is very general and it has many parameters. For the future work, we will implement several experiments and fix the sequences of operations or limit the random operations.

Budget

Considering the size of http requests in our budget, then none of the attacks can fit in the budget and the answer is no."

Take-Home Evaluation:

The challenge contains an intended vulnerability due to a race condition during defragmentation. An attacker can cause a while loop in backupInode to not terminate, consuming disk resources with each iteration.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In addition to identified vulnerabilities by us, collaborating with Iowa State University, Northeastern University and Gramma Tech, we identified another vulnerability in FileTable.backupInode() where an infinite loop can be triggered in a race condition, when a discrepancy occurs between filesystem on disk and the memory (sourceBytes and backupRegion). FileTable.backupInode() is only called in FileSystem.defragment() when the value of cleanMove is True and this happens when the filesystem is fragmented.

Therefore, to exploit this vulnerability, the attacker first needs to create a fragmented filesystem by for example, creating some small files and then updating them with larger files. Then, to trigger the race condition in FileTable.backupInode(), the attacker should have several concurrent threads running defragmentation.

KelinciWCA was not successful identifying this vulnerability because the race condition could not get triggered the infinite loop by sequential defragmentation calls. This question gives us some ideas on to improve our tools to adapt the fuzzer for multithreaded settings."

Live-Collaborative Evaluation:

Vanderbilt identified the intended vulnerability through collaboration. Additionally, the team noted a deficiency in their KelinciWCA fuzzer in handling race conditions.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9 Two Six Labs (Control Team)

A.5.9.1 Two Six Labs Overview

Two Six Labs answered 28 questions during the engagement with the third highest take-home accuracy of 71% and the 5th highest collaborative accuracy of 89%, despite being limited on which questions they could collaborate on. Two Six achieved a 100% accuracy in cases where they changed answers between the take-home and collaborative engagements.

During the take-home engagement the team had the most difficulty with AC Space questions and "No" responses achieving accuracies of 56% and 50% respectively. Following the collaborative engagement their accuracy on answered questions across all categories was ≥ 75% and their "No" accuracy was 100%.

A.5.9.1.1 Take-Home Engagement Scores

Table A-113: Engagement 5 Two Six Labs Take Home Engagement Question Accuracy

Question Type	Number Attempted	Number Correct	Accuracy (%)	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	7	88	5	62
SC Time	6	6	100	5	83
SC Space/ Time	0	0	0	0	0
AC in Space	9	5	56	5	56
AC in Time	5	5	100	5	100
Total	**28**	**23**	**82**	**20**	**71**

Table A-114: Engagement 5 Two Six Labs Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Number Correct	**Accuracy (%)**	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	23	18	**78**	15	**65**
Not Vulnerable	5	5	**100**	5	**100**
Yes Answer	18	18	**100**	15	**83**
No Answer	10	5	**50**	5	**50**

A.5.9.1.2 Live Engagement Scores

Table A-115: Engagement 5 Two Six Labs Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	6	75
SC Time	6	5	83
SC Space/ Time	0	0	0
AC in Space	9	9	100
AC in Time	5	5	100
Total	**28**	**25**	**89**

Table A-116: Engagement 5 Two Six Labs Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	**Reviewed Accuracy (%)**
Vulnerable	23	20	**87**
Not Vulnerable	5	5	**100**
Yes Answer	23	20	**87**
No Answer	5	5	**100**

A.5.9.2 Two Six Labs Responses

A.5.9.2.1 Accounting Wizard

A.5.9.2.1.1 Question 011 (SC Time, Intended Not Vulnerable, Take-home: Yes)

Take-Home Response:

“Yes, there is a side-channel vulnerability in time from which an attacker with Employee access can determine the project expenditure limit for a project they work on.

For a proof-of-concept, run the server via the `startServer.sh` script included with the program. Run the `exploit.py` python script from a python environment with the `requests` library installed, providing the hostname of the system the server is running on as the first argument and the name of the project to target with the side-channel attack as the second argument. (Run the `exploit.py` script with no arguments for full help documentation.) The script will log in as an

Employee, use Active Operations to add 26 items to the target project, use a combination of Active and Passive Operations to binary search the range of possible project expenditure limits, then use Oracle Queries to find the exact expenditure limit once the range has been sufficiently narrowed. (NB: the script performs Passive Operations and Oracle Queries by asking the user to supply the relevant information as needed.)

The vulnerability exploited here is the same vulnerability detailed in the answer for Question 023. In short, if a project contains many item budget lines, at least two of which are over half the project expenditure limit but less than the project expenditure limit, then producing a project expenditure report for that project will take an exponential amount of time. This vulnerability is used to induce a side channel. By adding a large quantity of items with costs so low that they will be included in the budget calculation no matter the project's budget and then two more items at a target cost, we turn the amount of time taken to produce an expenditure report into a 1 bit signal: if the expenditure report takes a long (i.e. exponential time), then the two items at the target costs were included in the calculation, so the project expenditure limit must be at least the target cost; if the expenditure limit returns quickly then the two items at the target cost were not included in the calculation, so the project expenditure limit must be below the target cost. From this primitive we build a binary search, picking a target cost in the middle of the range we are searching (starting with the project expenditure limit range of 1000-3000000 that is hardcoded into the program) and adjusting either the top or bottom of the range based on whether the project expenditure limit is above or below the target cost.

This process takes 5 Operations per iteration (2 Active Operations to add the two items at the target cost, 1 Passive Operation to measure the time taken to generate an expenditure report, and 2 Active Operations to remove the two items at the target cost) and will not require more than ~100 operations, since `log_2(3000000-1000) < 24`. Since there is a resulting surfeit of Operations, once the range has been reduced to <100 the script uses Oracle Queries to brute-force the project expenditure limit. As an Oracle Query only returns whether the true project expenditure limit is within $10 of a guess, the script begins by guessing $10 above the top of the current range and reduces its guess by 1 after each failed Oracle Query. Thus, the first Oracle Query to return `yes` will be for a guess exactly $10 above the true project expenditure limit, thereby revealing the project expenditure limit."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. There was a mistake in packaging this challenge program: the application came provided with no canned data (database of existing projects with varying content i.e. project expenditure limits, employees, etc.). While none of the side channel questions require this, it is a component of STAC that challenges have canned data. A clarification was provided during the live engagement that Blue teams can assume a database of existing projects, but there were no assumptions given on the projects' contents or states.

The reported side channel leaks whether the addition of a number of hours causes the project expenditure limit to be exceeded. The AC space vulnerability in logging is used to amplify the strength of this side channel. The EL believes that this reported side channel is present in the challenge program. The challenge question asks about leaking the project expenditure limit (PEL). Given a project under attack, there may be a number of items/hours charged to the project by other employees. The remaining amount can be referred to as the adjusted expenditure limit (AEL) where $PEL - other\ employees'\ hours\ and\ items = AEL$. We believe that this reported side channel leaks the AEL. Under the assumption that there are no hours/items charged to this

project by anyone other than the employee whose credentials are used by the attacker, this side channel leaks the PEL (secret). We do not believe that an employee-level user can set the hours/items charged by other users to zero. The minimum value of the secret is $1,000 and the maximum value is $3,000,000. The hours/items charged by other employees can have a range of $>$ $300,000. Without a way for an employee-level user to set the hours/items charged by other users to zero, this side channel cannot be used to leak the secret.

Several STAC teams identified an unintended vulnerability during the live collaborative engagement. In the case where an employee has no hours, an attempt to add hours causes an exception to be thrown. In handling the exception, the newly submitted hours are directly compared to the secret Project Expenditure Limit in FileStore.hoursFit() method. This comparison triggers writes to the log file if the newly submitted hours are less than or equal to the secret Project Expenditure Limit. The number of log writes that occurs controls the strength of the side channel. As delivered, the number of log writes is insufficient to distinguish when newly submitted hours exceed versus don't exceed the secret Project Expenditure Limit. However, an intended AC Space vulnerability where verbose logging is triggered can be combined with the addition of more items to a project to increase the number of log writes.

Post-Take-Home Analysis Ruling: Incorrect

A.5.9.2.1.2 Question 014 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"No. There is no algorithmic complexity vulnerability in space that causes the challenge program to store a file with a logical size exceeding the resource usage limit given in the input budget. While the program is rather chatty with logging and thus can be induced to print out relatively large amounts of log messages to the single log file with small requests (an effect amplified by switching the locale and thus the logging messages to Russian, as the character encoding used requires more bytes than the default English log messages) the resulting amplification is insufficient to overflow the resource usage limit given the operating budget. (ie, several hundred bytes of SSL handshaking and HTTP message produces only a few hundred bytes of logs.)

The FileStore storage used by the program is not attackable as each item purchase and set of Employee hours is stored in its own file. The file for each project contains a pointer to each item purchase and hour record for the project, which allows for arbitrary file size increases, but each new record under the project only increases the size of the project's file by 44 bytes (40 bytes for the ID of the record + 4 bytes of padding & metadata). As creating a new item purchase record under a project takes several hundred bytes, this is obviously not a feasible attack under the given input budget. (Employee hour records cannot be used because only one such record is created per Employee per project.)

If, however, the input budget counts only HTTP request bodies (rather than the entire SSL session) a successful attack is possible, as explained below:

For a proof on concept, run the server via the `startServer.sh` script included with the program and then run the `exploit.sh` script from the same computer, providing only the server IP/hostname as an argument. After executing `exploit.sh` the `.accounting.log` file used by the program for logging will have a logical size of 1222823 bytes as reported by `stat`. `exploit.sh` logs in (53 bytes PDU), sets the program locale to Russian (0 bytes PDU), then attempts to

record hours for a non-existant project 3074 times (3074 * 16 bytes PDU), for a total of 49237 bytes PDU, under the input budget of 50 kB.

The vulnerability lies in the error messages logged by the program when a user attempts to bill hours to a non-existant project. These error messages (see the various `*_ru_RU.properties` files) are rather long and so allow the user to amplify 16 PDU bytes of input into several hundred bytes of messages written to the log file at a ratio sufficient to exceed the resource usage limit within the input budget.

An even stronger amplification attack can be mounted if a Manager has created a project. When an Employee updates their hours for a project via the endpoint `/employee/updateHours/PROJECTNAME` ~1.7 Kb of logs are written at the expense of only 14 bytes of HTTP message (`{"quantity":1}`), producing an amplification factor of over a thousand, thus exceeding the resource usage limit in less than 600 requests, for a total PDU cost of < 8 Kb."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attempt to switch the local to and unsupported on triggers verbose logging mode that can be used to submit inputs that cause the server to exceed the resource usage limit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"[…] When the user sends a request to change the logging language `Logger.SetLocale()` is called. This function calls `Logger.supportsTag()` to check if the language requested by the user is supported by the program (ie is in the list 'en_US', 'en_UK', 'ru_RU'). If the user's requested language is not supported a `LocaleNotSupportedException` is thrown. The constructor for this exception, `LocaleNotSupportedException.LocaleNotSupportedException()`, sets the log level to `TRACE`, the most verbose log level.

Whenever the user successfully purchases an item under a project that project is written out to its file again by `BaseObject.write()`, which calls `BaseObjectMarshaller.toBytes()` on the `Project` object. `BaseObjectMarshaller.toBytes()` calls `BaseObject.uuid()` on each child (ie item purchase and hour charge) of the project. `BaseObject.uuid()` contains a TRACE-level log statement. Thus, each time an item is purchased under a project a log message will be printed for each item and hour record under that project. […]"

Live-Collaborative Evaluation:

Two Six Labs identified the intended vulnerability collaborating with other teams.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.1.3 Question 023 (AC Time, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"[…]The vulnerability lies in the algorithm the program uses to `solve` the budget for a project. When an Expenditure Report is requested, `ManagerRoutes` uses an `ExpenditureReportBuilder` to build an `ExpenditureReport` object, which is passed to the constructor for `JSONExpenditureReport`. `JSONExpenditureReport()` calls `getBudgetLines()` on the passed

`ExpenditureReport`, which, for each project in the report, calls `BudgetSolver.getAllowedExpenditures()`, passing the project and its Expenditure Limit.

`BudgetSolver.getAllowedExpenditures()` first subtracts all non-optional budget lines (ie employee hours) from the Expenditure Limit. It then calculates the total of all the optional budget lines (ie item purchases). If this total does not fit within the adjusted Expenditure Limit the `BudgetSolver` removes optional budget lines with costs greater than the Expenditure Limit and then calls `BudgetSolver.getAffordableLines()` in order to find a subset of the optional budget lines (ie item purchases) that will fit within the adjusted Expenditure Limit. This function sorts the optional budget lines in descending order and then generates subset of the item purchases by calling `BudgetSolver.vectorDec()`, checking each subset and returning the first one that fits within the adjusted Expenditure Limit. `BudgetSolver.vectorDec()` is vulnerable because it generates every possible subset of the input set of optional budget lines, the power set of the input set, which is exponential in the size of the input set. (The power set of a set with `n` items contains `2^n` subsets.) […]"

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.1.4 Question 026 (SC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"The vulnerability lies in the storage format used for projects. The program stores each project in an individual `chunk` file. Each project's budget lines are stored in memory as `children` of the project (for reference, see `FileStore.newItem()`, which calls `BaseObject.add()` to add the new item as one of the `children` of the project). When the project is written out to a file by `BaseObjectMarshaller.toBytes()`, each child is recorded as a 40-byte UUID (generated by `BaseObject.uuid()`) preceded by three bytes of `0x00` and one byte of `0x28`. Thus, whenever a new budget line is added to a project the size of the `chunk` file used to store that project will increase by 44 bytes. This information, while necessary for the side-channel attack, induces a vulnerability because the endpoint `/employee/report/files` returns a histogram of the sizes of the `chunk` files stored by the program.

Thus, the attack proceeds as follows: the employee requests a file size histogram and records it as the baseline, adds an item (with a random name, to ensure that it creates a new budget line) to the target project, and requests a second file size histogram. This second histogram is compared to the baseline. If a there is a file size in the second histogram that gained a file between the histograms and is exactly 44 bytes larger than a file size that lost a file between the histogram, it must be the file size of the target project. Having found the file size of the target project, the employee then divides it by 44 to obtain a candidate number of budget lines and submits this guess as an oracle query. If this guess is incorrect, due to the project's other metadata totaling over 44 bytes and thus inflating the apparent number of budget lines, the employee makes further oracle queries, decreasing the candidate number of budget lines by 1 until the true number is found."

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.2 BattleBoats 1

A.5.9.2.2.1 Question 006 (AC Time, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“The StrikeLocator.toRadians() method converts a degree measure to radians by first adjusting the degree to a value between 0 and 360. Instead of using a modulus operator, the code iteratively subtracts 360. The input to this function is the opponent's vertical angle and board angle parameters when taking a shot. The inputs are not sanitized prior to being processed by the toRadians() function, so an attacker can shoot with vertical angle theta, and cause the while loop in the toRadians() function to run for theta / 360 iterations. While the toRadians() function is running, the opponent will not be able to process any other requests, including the benign 'help' command. Sending a theta value greater than 15 trillion causes the runtime of the benign request to exceed 120 seconds. The `shoot 0 1 15000000000000 0` message sent to the opponent is only 81 bytes long. The runtime can be made arbitrarily large by sending even larger values for the vertical angle, e.g 1e999.”

Take-Home Evaluation:

Two Six Labs identified a known but unintended vulnerability in the challenge program.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.2.2 Question 028 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“A side channel vulnerability in space revealing the cannon location must access the cannon coordinates, and send a message to the attacker that would vary in size or content based on the cannon's location. The `print ocean` command, which prints the exact location of the cannon on a player's ocean grid, is not vulnerable to SC Space attacks since the attacker would have no way of triggering an opponent's `print board` command, nor would an attacker be able to see the results of the command as the grid is printed locally and nothing is sent over the network. We examined the decompiled source code, searching for all uses of the cannon location. We identified two instances where the cannon location is accessed in preparing a response to an oppnent's message.

First, in the EndPlacingShipsCommand, the BattleBoatsMessage.ReadyMsg sent to the opponent is this.warShips.rarefy(this.warShips.getPlacementMessage()). The WarShips.getPlacementMessage() function returns this.rarefy(this.currentCompetition.getPlacementMessage()). The Competition.getPlacementMessage() is interesting because it returns "Boats and cannon have been placed" concatenated with the result of this.cannonSquare.obtainEncoding(), which returns a string containing x + 23*y spaces for a connon placed at (x,y). Since the side length is never more than 20, x is always less than 23, and so the number of spaces in obtainEncoding() uniquely identifies the cannon location (x = length % 23, y = length / 23). Before the message is sent to an opponent, however, it makes two passes through WarShips.rarefy(), which takes a messages and returns a string containing the length of the message. After the first pass through rarefy(), the result is 33 + obtainEncoding().length(), which would still uniquely identify the cannon location. However, this value will range from a minimum of 57 for a cannon placed at

(1,1) to a maximum of 513 for a cannon placed at (20,20), so after the final pass through rarefy(), we will be left with a value of either "2" or "3". This does reveal some information about the cannon location. Specifically, if the value is "2", an attacker can deduce that the cannon has been placed in one of the first 2 columns, since for a cannon with y value 3 or larger, 33 + obtainEncoding().length() is at least 33 + 23*3 + 1 = 103, whereas the value is less than 100 for a cannon placed anywhere in the first two columns. If an attacker's cannon were to be placed in the first two columns, a maximum of 40 oracle queries would be required to guess the exact location, which is within the budget for this question. However, we must assume a worst case secret, where the attacker's cannon has not been paced in the first two columns. In this case, up to 360 oracle queries could be required to determine the cannon location. To identify the location with 95% success would require up to 342 active operations. We found no way to further reduce the search space using a space side channel, so determined that while this is a side channel, it is not strong enough for a player to discover the exact coordinates of their cannon on an opponent's board within the 100 operation input budget.

The second avenue we explored for possible space side channel's was the opponent's response to a player's shot. The cannon coordinates are used to translate between positions on the opponent's ocean board, and the attacker's radar board. We thought that maybe responses to an opponent's shot would be different based on whether a shot landed on or missed the opponent's ocean board. We noted that OceanBoard.grabOceanSquares() does check for shots that miss the ocean board (line 117 in Luyten) by checking !this.onOceanBoard(pixelX) || !this.onOceanBoard(pixelY), but only returns an empty HashMap if the shot also would miss the radar board as determined by this.tooFar(xGap) || this.tooFar(yGap). Sending input that would go down this code path is not useful, because shots from the attacker that are off the radar board would not provide any information about the cannon location. The OceanBoard.fetchStrikes() function correctly collects all squares that would be hit on the radar board, not just the ocean board, so we concluded that there is not a side channel in space from shooting at an opponent's board.

Having examined all locations in the source code that reference the cannon location, we conclude there is no side channel vulnerability in space from which an attacker can determine the exact coordinates of their own cannon on an opponent's ocean board within 100 operations."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.3 BattleBoats 2

A.5.9.2.3.1 Question 010 (SC Time, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“When the WarShipsDispatcher receives a shot message from the opponent, it calculates the hit squares relative to the ocean panel, and then translates this to hit squares and results of the shot on the opponent's radar panel. In the OceanPanel.pullOceanSquares() method (line 117 using Luyten decompiler), this.takeHits() and this.updateHitsAsSunk() are called twice if (!this.onOceanPanel(pixelX) || !this.onOceanPanel(pixelY)), i.e. if the opponent's shot missed the board, but only once for shots that hit the board. An attacker can strategically place shots, and compare the opponent's response time for the shots. Whichever shots had the shortest response time landed on the board, while the shots with the longer response time missed the board.

To exploit this side channel, an attacker can take shots at radar positions (-1,0), (0,1), (0,2), ... , (0,n) along the y-axis, and also at positions (0,-1), (1,0), (2,0), ... , (n,0) along the x-axis, where n is the board size. The attacker should measure the amount of time between sending a shot request and receiving the response from the opponent. For the shots along the y-axis, look at the difference in response times from a shot at (0,i) to (0,i+1) (or (0,i+2) for -1 to 1), and order this list of timing deltas along with the corresponding shot from largest to smallest. Only include differences in which the response time increased, since we are looking for a board edge after which we expect the time to increase. Response times for shots that land on the opponent's board should be close to each other and shorter than response times for shots that miss the board, so the difference in timing between shot (0,i) which hit the board and (0,i+1) which missed should be at or near the top of the list. Repeat this process for the x-axis. If the largest deltas occurred between shots (0,i), (0,i+1) on the y-axis, and (j,0), (j+1,0) on the x-axis, then it is likely that the top and right edges of the opponent's ocean board, so the most likely cannon location is (n-j,n-i). Note that (0,0) is always on the ocean board, and (0,n) and (n,0) are always off the ocean board, so there will always be pairs (0,i), (0,i+1) and (j,0),(j+1,0) with 0 <= i,j < n such that the first shot hits the opponent's board and the second shot misses the board.

Assuming a worst case scenario where the board is of size 20, timing the initial shots require 42 active operations. This leaves 58 operations for oracle queries. In case the timing data is not perfect and the edge of the board does not have the largest timing difference, we can query all possible combinations of the 7 most likely x and y coordinates based on the sorted delta lists (up to 49 oracle queries) while staying within the input budget. In our testing, this margin for error was sufficiently large to reliably determine the exact location of the cannon.

We used the shot_help command to set parameters for shots. The shot_help values keep the height and launch angle constant, varying only the velocity and board angle. We chose to use shots at (-1,0) and (0,-1) instead of (0,0) to keep the shot parameters as uniform as possible. In our trials, the difference in timings for different launch parameters given by shot_help did not mask the timing side channel enough to prevent us from disambiguating which shots landed on or off the opponent's board.”

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.3.2 Question 021 (AC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“There is a loop in the HitLocator.computeHitTime() method which, after 25 iterations, begins adding entries to the trajectory Map on each successive iteration. This code is called when an

opponent makes a shot. We determined experimentally that the number of iterations through this loop can be made very large by sending a height of 0 and a very small initial velocity. An attacker can drive the memory usage of an opponent above 1.5 GB by issuing `shoot 0 1e-500000 0 0`. The size of the shoot message sent to the opponent is only 76 bytes. We also note that this doubles as an AC Time vulnerability, as the runtime of computeHitTime() increases as the number of iterations through the loop increase, during which commands from the benign user will not be processed. The memory usage (and runtime) can be increased by making the initial velocity even closer to 0, e.g. 1e-9999999999."

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.4 BraidIt 1

A.5.9.2.4.1 Question 002 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"No. There is no timing side-channel under the operational definition. All of the critical communications between two braidit clients takes place at the convenience of those individual users. For instance, a user chooses when to offer a game, accept a game, choose braid lengths, send braid lengths, modify their received braids, and send the final braids to their opponent. There is no automated response mechanism to time packet responses that would reliably leak any braid data.

However, there is an attack in which a user can correctly guess the modified braids using zero active operations or oracle queries. This attack relies on an un-intended vulnerability, which we describe here. Since the attacker is assumed to be a player in the game, the attacker can modify their own client. The braid validation mechanism is performed entirely on the side of the guessing party. Because all five unmodified braids, as well as the modified braid are received by the guessing party, the attacker can run the validation on each unmodified braid before submitting a formal guess. Because the attacker is a participant in the game, all of the braids received can be decrypted and read in the clear.

The validation function is contained in the Plaitit class, and is called "isEquivalent". Validation proceeds by first checking whether the number of fibers is the same between the two compared braids (for simplicity call them braids A and B). Next Braid B is "inverted" where it is reversed and all of the cases of the braid characters are switched (e.g. dEF->feD). The inverted braid B is concatenated with braid A. During the validation process, the concatenated braid is scanned for pairs of characters differing by case. If a pair is found, then operations are performed to bring those two characters together (that mirror the valid braid modifications). When such a pair is eventually made adjacent, these characters are canceled out, reducing the size of the concatenated braid. Eventually, you are left with a string with no pairs or the empty string. In the first case, the two braids A and B are not equivalent, and in the second case they are equivalent. Since this entire process only depends on the plaintext content of the braids, a client with access to the braid plaintext can perform this validation step at their leisure for each of the five received braids, effectively winning any contest without using any active operations (aside from those required for playing a normal game) or oracle queries."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. Two Six Labs identified a potential vulnerability to allow an attacker to win the game. Note this is a distinct task from determining the selected and modified braid. There are however attacker-controlled cases where the two are equivalent.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“During the Live Engagement, Colorado noted there is an attack involving the timing difference between submitting the make_guess command and sending the notification of who won a round to the opponent. To setup the attack, the attacker should set the number of strands to 3, which disables all modifications to the braid that change the length of the braid. Then the attacker can set braid lengths to the extreme lengths (e.g. two to 1, three to 50). Colorado noted that there is a measurable timing difference between evaluating isEquivalent() for braids of length 1 and 50, which allows the attacker to identify which length braid was modified by the opponent.”

Live-Collaborative Evaluation:

The challenge program does not contain an intended vulnerability. Following the “make-guess” command Two Six reported a potential timing vulnerability. This is a timing vulnerability on the attacker’s own application instance, and STAC side channel vulnerabilities for remote applications (server-client, and peer-to-peer) are on a remote server or remote peer under attack. As the attacker is free to modify their own application, the attacker does not need a side channel to view any information that can be plainly viewed in that application.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.4.2 Question 007 (SC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“There is a side-channel in space for this program. Since the attacker initiates the game, they can set the lengths of each of the candidate braids. All of the legal braid modifications (expand3, expand5, swap and flipRandomTriple) maintain whether the braid is of even or odd length (expand3 adds 2 characters, expand5 adds 4 characters, swap and flipRandomTriple do not change the length). Therefore, after any number of modifications, the chosen braid will remain either even or odd, whichever it started as.

The attack works as follows: player 1 starts the game and sends over lengths of half even and half odd (since there are 5 braids, choose 3 to have even lengths and two to have odd lengths). When the target responds, the attacker checks whether the received string has even or odd length (1 active operation used so far). Then the attacker guesses from the available options based on the parity (at worst case the string is of even length and there are 3 choices). If the attacker guesses incorrectly, then there are two remaining possibilities for the braid in round 1 (2 active operations used so far). Fast forward to round three, the attacker uses the same strategy (3 even length braids, two odd length braids). Again at worst there are 3 possibilities once the attacker receives a response (3 active operations used so far). At this point, in the worst case there are two possiblities for the braid from round 1 and three for the braid from round 3, leaving 6 total possibilities. This falls within the input budget of 9 remaining oracle queries.

However, there is also an attack in which a user can correctly guess the modified braids using zero active operations or oracle queries. This attack relies on an un-intended vulnerability, which we describe here. Since the attacker is assumed to be a player in the game, the attacker can modify their own client. The braid validation mechanism is performed entirely on the side of the guessing party. Because all five unmodified braids, as well as the modified braid are received by the guessing party, the attacker can run the validation on each unmodified braid before submitting a formal guess. Because the attacker is a participant in the game, all of the braids received can be decrypted and read in the clear.

The validation function is contained in the Plaitit class, and is called "isEquivalent". Validation proceeds by first checking whether the number of fibers is the same between the two compared braids (for simplicity call them braids A and B). Next Braid B is "inverted" where it is reversed and all of the cases of the braid characters are switched (e.g. dEF->feD). The inverted braid B is concatenated with braid A. During the validation process, the concatenated braid is scanned for pairs of characters differing by case. If a pair is found, then operations are performed to bring those two characters together (that mirror the valid braid modifications). When such a pair is eventually made adjacent, these characters are canceled out, reducing the size of the concatenated braid. Eventually, you are left with a string with no pairs or the empty string. In the first case, the two braids A and B are not equivalent, and in the second case they are equivalent. Since this entire process only depends on the plaintext content of the braids, a client with access to the braid plaintext can perform this validation step at their leisure for each of the five received braids, effectively winning any contest without using any active operations (aside from those required for playing a normal game) or oracle queries."

Take-Home Evaluation:

The challenge contains an intended vulnerability. An attacker can use the encrypted packet sizes to determine the secret. The first vulnerability reported by Two Six Labs does not appear to properly determine the cost of active operations. Active operations are counted via network requests to the remote end-point not commands to the console. Under these restrictions the attack outlined by Two Six Labs exceeds the operational budget. The second vulnerability flagged by Two Six Labs assumes that winning the game is the same as determining the braid selected and modified. There are attacker input sequences for which this is true, but Two Six Labs does not appear to have discovered/noted these in the response.

Post-Take-Home Analysis Ruling: Incorrect

A.5.9.2.4.3 Question 017 (AC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"The Plait.isEquivalent() method is used to determine whether two braids are equivalent. It calls the simplifyCompletely() method, which launches a while loop successively calling simplifyOnce() and freeSimplify() until isSimplified() evaluates to true. Within freeSimplify() and simplifyOnce(), there are logging statements written to log files. The freeSimplify() function removes pairs of adjacent characters if they are the same letter, but one is uppercase and the other is lowercase. The simplifyOnce() function identfies pairs of opposite case letters via obtainEnsuingPortion(), and moves them within the string so that freeSimplify() will remove them. However, the obtainEnsuingPortion() function skips over pairs of matching letters if they are more than 25 characters apart. In JD-GUI, this is on line 452 of Plait.class in the decompiled source code. By checking the equivalence of the alphabet as one braid, and the reversed alphabet

as the second, the meetings string begins as ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz, which cannot be reduced by freeSimplify() or simplifyOnce() since all opposite case letter pairs are 26 characters apart. The isSimplified() function continually returns false, which keeps the while loop running. Submitting these two braids to the equivalence check triggers an infinite loop, which writes out over 40 MB to a set of 4 log files within the staclog directory."

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.5 BraidIt 2

A.5.9.2.5.1 Question 001 (SC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"There is a side-channel in space for this program. Since the attacker initiates the game, they can set the lengths of each of the candidate braids. All of the legal braid modifications (expand3, expand5, swap and flipRandomTriple) maintain whether the braid is of even or odd length (expand3 adds 2 characters, expand5 adds 4 characters, swap and flipRandomTriple do not change the length). Therefore, after any number of modifications, the chosen braid will remain either even or odd, whichever it started as.

The attack works as follows: player 1 starts the game and sends over lengths of half even and half odd (since there are 5 braids, choose 3 to have even lengths and two to have odd lengths). When the target responds, the attacker checks whether the received string has even or odd length (1 active operation used so far). Then the attacker guesses from the available options based on the parity (at worst case the string is of even length and there are 3 choices). If the attacker guesses incorrectly, then there are two remaining possibilities for the braid in round 1 (2 active operations used so far). Fast forward to round three, the attacker uses the same strategy (3 even length braids, two odd length braids). Again at worst there are 3 possibilities once the attacker receives a response (3 active operations used so far). At this point, in the worst case there are two possiblities for the braid from round 1 and three for the braid from round 3, leaving 6 total possibilities. This falls within the input budget of 9 remaining oracle queries.

However, there is also an attack in which a user can correctly guess the modified braids using zero active operations or oracle queries. This attack relies on an un-intended vulnerability, which we describe here. Since the attacker is assumed to be a player in the game, the attacker can modify their own client. The braid validation mechanism is performed entirely on the side of the guessing party. Because all five unmodified braids, as well as the modified braid are received by the guessing party, the attacker can run the validation on each unmodified braid before submitting a formal guess. Because the attacker is a participant in the game, all of the braids received can be decrypted and read in the clear.

The validation function is contained in the Weave class, and is called "isEquivalent". Validation proceeds by first checking whether the number of fibers is the same between the two compared braids (for simplicity call them braids A and B). Next Braid B is "inverted" where it is reversed and all of the cases of the braid characters are switched (e.g. dEF->feD). The inverted braid B is concatenated with braid A. During the validation process, the concatenated braid is scanned for

pairs of characters differing by case. If a pair is found, then operations are performed to bring those two characters together (that mirror the valid braid modifications). When such a pair is eventually made adjacent, these characters are canceled out, reducing the size of the concatenated braid. Eventually, you are left with a string with no pairs or the empty string. In the first case, the two braids A and B are not equivalent, and in the second case they are equivalent. Since this entire process only depends on the plaintext content of the braids, a client with access to the braid plaintext can perform this validation step at their leisure for each of the five received braids, effectively winning any contest without using any active operations (aside from those required for playing a normal game) or oracle queries."

Take-Home Evaluation:

Two Six Labs provided the same response as for Question 007. The same conditions on that response apply here

Post-Take-Home Analysis Ruling: Incorrect

A.5.9.2.5.2 Question 022 (AC Time, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"The Weave.isEquivalent() method is used to determine whether two braids are equivalent. It calls the reduceCompletely() method, which has a while loop that does not terminate until isNormalized() evaluates to true. The isNormalized() function walks through this.intersections and finds the lowest remaining character, then checks to see if the inverse character is also in the string. This check is performed twice, once each with local variables `lowest`, and `base`. The only difference is that while lowest is initialized to 65535, base is initialized to (char)(122 - this.penalty + 1). The penalty value, which is set by calculatePenalty(), is 0 if there are fewer than 26 unique characters in the braid, and otherwise it is the difference in number of occurrences between the most and least common characters in the braid. Since the maximum braid length is 52, we can make the penalty as large as 26 with a braid that has 25 different characters occurring once, and one character occurring 27 times. This means the base character can be initialized to anything between char(122 - 0 + 1)='{', and char(122-26+1)='a'. We realized that if base were initialized to 'a', it would never be overwritten because Character.toLowercase(c) will never be strictly less than 'a'. If additionally this.intersections contains 'A', then this.intersections would contain the inverse of base, so isNormalized() will evaluate to false. Finally, if the braid also does not contain 'a', then the 'A' will not ever be removed from the braid via calls to reduceCompletelyEntity(), so subsequent calls to isNormalized() will continue to return false, triggering an infinite loop.

Our exploit shows a proof of concept of a braid (AAAAAAAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOPQRSTUVWXYZ) which contains 'A', does not contain 'a', and has a penalty of 26 (27 A's, 1 occurrence of B-Z). The exploit will trigger an infinite loop when run with any braid that does not contain at least 27 'A's (the second braid gets inverted, so the 'A's become 'a's). In order to affect the benign user, an attacker using a modified client can initiate a game with player B with 27 strands. The benign user accepts the game, and plays the first round as normal with the attacker setting lengths, the benign user modifying a braid, and the attacker making a guess. In round 2, player B sends lengths to the attacker. The attacker selects random braids matching the lengths as usual. But then rather than modifying one of the braids, the attacker's modified client sends the malicious modified braid

'AAAAAAAAAAAAAAAAAAAAAAAAAAABCDEFGHIJKLMNOPQRSTUVWXYZ'. One way to do this is to modify WeaveElectedState.grabWeaveString() to set weaveStr (line 35 in JD-GUI) to the malicious braid string instead of this.weave.pullIntersections(). The modified client could also ensure that none of the randomly selected braids contain 27 'A's, although the chance of this happening randomly would be extraordinarily rare. It is now player B's turn to guess which braid has been altered. Whichever braid player B selects will be compared against the modified braid and trigger the aforementioned infinite loop, after which any 'help' command issued by player B will never receive a response."

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.6 IBASys

*A.5.9.2.6.1 Question 015 (SC Space, Intended Vulnerable, Take-home: *Yes, Live: *Yes)*

Take-Home Response:

"No, there is no side channel vulnerability in space. In order to impersonate the target user, the attacker would need to learn either that user's passcode, that user's authentication image, or the session code returned by the server to that user upon authenticating. There is no leak in space in either the client or the server portions of the program that reveals any of these items.

The user's authentication image and passcode cannot be directly leaked, as the attacker has no ability to observe either via passive operations. Thus, the attacker must utilize only the single authentication endpoint exposed by the challenge program server.

The session code sent back by the server upon successful authentication is always sent back in a block whose size is not dependent on the value of the session code (the block is 6 bytes "result", 32 bytes for the session code, then the user name). An identically sized block is sent in response to a failed authentication (the block is identical to the block sent for a successful authentication, but with a random session code). Thus, the size of the response from the server to any active operation by the attacker does not leak any information about the target user's passcode, authentication image, or session code, as the only space dependent part of the response is the user name, which is in no way privileged information. The attacker can learn whether or not an authentication image is the target user's authentication image by sending it to the server twice and checking to see if the session codes returned are identical (indicating a correct guess), but this is not a side-channel in space and the search space of possible passcodes is far too large for such a brute-force attack to be viable. Thus, the sizes of the packets produced by the server do not leak any information and there is no side channel vulnerability in space.

We did identify a way to learn information which can be used to impersonate the target user, however our attack does not require use of a side channel in space. The attack works as follows:

1) The attacker sends the first login request packet with a header containing the attacker's valid username and the first payload of the attacker's valid passcode image

2) The attacker sends all subsequent login request packets with the target user's username and the remaining payloads of the attacker's valid passcode image

3) The server will respond with the target user's legitimate user token.

This attack works because the server looks up the secret passcode to verify against upon receiving the username contained in the initial packet header but uses the username in the final header received to retrieve the secret token if the passcode is validated. Thus, an attacker sending their own username initially can send their own image and it will be validated against their passcode, and by sending a target user's username in subsequent packets the attacker will receive the target user's secret token in response. While the attacker never learns the target user's passcode, our interpretation, bolstered by the death ray warning in IBASysClientParser, is that the secret token is information which can be used to impersonate the target user.

We did not consider this attack a side channel in space because it requires no monitoring of the number, content, or length of network packets sent between the client and server. In order for the attack to succeed, an attacker only needs to modify their client to send out packets with the headers as described."

Take-Home Evaluation:

Two Six Labs did not identify the intended vulnerability by which a race condition can be used to determine the passcode string bit-by-bit. This string can then be used to generate an image which will be accepted by the application. The challenge question asks if the attacker can learn information that allows them to impersonate the target user. This application only allows users to submit inputs for authentication after which a token is returned. Two Six Labs's vulnerability which leaks the token via a channel (not a side channel) allows the attacker to perform an action only a validated user should be able to perform i.e. impersonate that user. To maintain consistency with other cases of reported out-of-scope vulnerabilities, this response of "No" has been changed to "Yes" and given a post-engagement analysis evaluation of correct.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"[…] Additionally, per the clarification provided during the Live Engagement, this attack does not meet the criteria for gaining information used to impersonate the target user because leaking a target user's token does not allow the attacker to perform any actions as the target user within the context of the IBASys application."

Live-Collaborative Evaluation:

Impersonating another user is to span the set of possible outputs from the server for all authenticated user requests. This application's only output for all authenticated user requests (only request is a username + passcode image) is to produce a (token) for that authenticated user. Therefore, as this attack accesses the only output that an authenticated user can receive, this response is changed to an out-of-scope "Yes" and given a post-engagement ruling of correct.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.6.2 Question 024 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"We first note that the IBASys server returns only one message in response to a request, so the only avenue for an SC Timing attack must be to vary the input image in a way that causes the timing of the response packet to change in a way that reveals information about the target user. After examining the decompiled source code, we identified only one place where processing time would be affected by the input image in a way that could reveal bits of a target user's

passcode. Based on our examination of the source code, evaluation of validation of a user's image proceeds roughly as follows:

1) the image is cropped to just the 100x100 RGB image in the upper left corner,

2) the 100x100 image is flattened into an array of 10000 integers representing the color of each pixel in the cropped image,

3) every 78th entry in the array is added to a new array modulo 2, creating a 128-bit key,

4) every 4th item in the key is compared modulo 2 to the user's stored passcode, which is an array of 128 integers.

If the provided image passes this check in step 4, the legitimate user token will be sent in the response message, and a random token will be returned otherwise. It is at step 4 that we identified a side channel in the code. The code in question is the for loop which does the comparison of the image derived key and user passcode, which is in IBASysServer.ImageMatcherWorker.run() on lines 292-304 in JD-GUI. The issue is that line 301 contains a break statement which is triggered if the key fails to match the passcode, so this loop will run longer if more parts of the key match the secret passcode.

Theoretically, an attacker could provide two images that are identical except for the first pixel being either white or black, see which took longer to run, and could use this to determine the first entry of the passcode modulo 2. The attacker could then continue with two more images whose derived keys are identical except for the second element, and so on until the attacker learns the correct key in its entirety and can produce an image with which to impersonate the target user. In practice, however, the side channel is weak for several reasons. First, it is a very tight loop, iterating through at most 32 elements, and doing only a few inexpensive computations, so even with timing information for this loop in isolation, a large degree of precision would be required to identify differences in the number of iterations. Second, as mentioned previously, the only timing granularity available is from when the server receives the final image packet to when the server issues a response, which includes other operations with runtime that is sufficiently variable to mask the precise timing information needed to exploit this side channel. To confirm our belief that this side channel was not strong enough, we created a set of 33 images of identical size and differing only in the pixels that impact the key. The images progressed from matching 0 up to the full 32 elements of a target user's passcode. We tried to further reduce variation by modifying the client to select the same IV every time through the program. We timed the response from the server 50 times for each image. In order for an attack to succeed we would need the response times to be strictly increasing from the image matching 0 elements to the correct image. As expected, upon analyzing the results, no such order emerged. We note that 3500 operations would allow for up to 109 trials per image, however it was clear to us from the initial test with 50 that even with another 50 trials the images were not going to converge to a neatly ordered list. For example, the correct image, which would need to take the longest amount of time, sat squarely in the middle of our result list. Our conclusion, then, is that there is no side channel in time from which an attacker can learn information which allows them to impersonate the target user within the 3500 operation input budget."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

The application does not contain an intended in-scope SC Time vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.6.3 Question 029 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“No, there is no algorithmic complexity vulnerability in space that would allow an attacker with a valid IBASys username to cause the challenge program to store files with combined logical sizes that exceed the resource usage limit.

The challenge program, whether operating in client or server mode, does not write any data to the filesystem at any point and so filesystem writes cannot be exploited to exceed the resource limit. The methods "AbstractScalrTest.save()", "CryptoUtils.enCrypto()", "FileEncryption.encrypt()", and "FileEncryption.saveKey()" do write files but are never called. The program does contain many logging statements that would write to a logfile (via "Scalr.log()"), but almost all such statements are wrapped in "if" statements that perform the "Scalr.log()" calls only if the system property "imgscalr.debug" is true. This system property is set to false by default and while several functions in "ScalrResizeTest" set "imgscalr.debug" to true, none of the code in "ScalrResizeTest" is called or accessible from anywhere else in the program. Thus any "Scalr.log()" wrapped in checks for "imgscalr.debug" will never activate.

There are several calls to "Scalr.log()" that are not wrapped in such "if" statements, but all such calls pass "null" or an array of "null" objects as the third argument to "log()" (in "IBASysServer.init()", the penultimate "catch" statement calls "Scalr.log()" and passes "new Object[] {session}" as the third argument, but an exception can only be thrown in the corresponding "try" block before "session" is initialized to a non-"null" value, since the lone call after "session" is initialized is to "LoginManager.parsePacketPart()", which will not throw any uncaught exceptions). "Scalr.log()" will only write to log files if the first object in the third argument is non-null, so these calls do not result in the program consuming space in any file.”

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The persistence of a session object can be used to submit a malformed request that bypasses the input guard to trigger an ArrayIndexOutOfBounds exception causes excessive logging to disk.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“There is a call to Scalr.log() in an exception handler within IBASysServer.init(). The call does not check whether debug mode is turned on. It will log the session object to a file if an exception is triggered in LoginManager.parsePacketPart(). We had analyzed this function previously for the potential to write out data to the log, but did not realize that it was possible to trigger an ArrayIndexOutOfBoundsException. The exception can be triggered by sending a payload consisting entirely of null bytes. A section of parsePAcketPart() looks for the first non-null byte, and tries to access data1[-1] when sent the malicious payload.

The initial log file written due to the exception is not larger than the input budget, but by changing the username in the header with each packet sent, new, larger log files will be written each time. Sending 500 KB worth of null bytes in 50 packets, which requires less than 1 MB of input, will drive the combined size of the log files over 200 MB.

Modifying the client is necessary to change the user name for each header and send the null payload. Colorado explained and ran their exploit script during the live engagement; other teams had similar exploits. We were able to separately verify their observations."

Live-Collaborative Evaluation:

Working with Colorado, Two Six Labs identified the intended vulnerability.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.7 Medpedia

A.5.9.2.7.1 Question 009 (SC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"No, there is not a side channel in space that allows an attacker to discover the article requested by another user. Similar to the case with the poker SC in space, there is random padding that appears to thwart any attempt at this side channel. When an article is requested, the getMasterPage() function in the ContentController class is triggered. This method actually fetches the page to send, however, it also adds a random amount of padding bytes between 32 and 2048. This is done through a Java SecureRandom call. This padding is enough to prevent the side channel in space under the operational budget.

To confirm that the padding prevents the sidechannel, a simulation script was written. Similar to the case for the poker simulation script, this script makes incorrect assumptions that are in favor of the attacker, to further ensure that the side channel is not possible (for example, the script assumes that the actual filesizes of the articles are seen over the wire. In reality, these will be under SSL/TLS, where the size over the wire will likely be binned into a multiple of a cipher block size, making the side channel more difficult). The script and supporting data file are provided as "simulation.py" and "filesizes.txt".

One interesting note about this program is that many methods of SecureRandom are overwritten with custom code, such as the SHA-1 PRNG. This was investigated, as if we could predict the random numbers being generated, then we could negate any padding added to the article, enabling the side channel. However, we could not find any vulnerabilities in the custom crypto code that would allow us to predict the random padding size."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The random call does not have sufficient entropy. The attacker can determine the possible paddings using active operations and use the relationship of html page size to article to determine the secret. Two Six Labs noted correctly that the ability to predict the random padding is essential for an SC vulnerability, but did not observe that the random padding is sufficiently restricted.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Yes, there is a side channel in space that allows an attacker to discover the article requested by another user. This side channel is in the size of article sent to the benign user over the wire. Similarly to the case with the poker SC in space, there is random padding that is added to client messages that appears to thwart any attempt at this side channel. However, the random padding is weak. On server startup, 8 possible values of random padding length are determined (these change every time the server is started). By using active operations to perform a coupon-collector problem (i.e. see all 8 possible random padding values with high probability), we can use that to help identify the article based on its size on the wire. Using 59 active operations is more than enough to obtain all 8 values with over 99% confidence, so we use this to be safe.

When an article is requested, the getMasterPage() function in the ContentController class is triggered. This method actually fetches the page to send, and then adds one of the 8 possible values of random padding. Our strategy is as follows: Use roughly 59 active ops to determine the 8 possible random padding values. Then, when the values are determined, we use a passive op to see the article being sent from the server to the client. While there are no other fields sent in the HTTP data besides the article, there is padding on the packet data due to SSL/TLS cipher block size, which in the worst case causes the packet data size to be rounded up to the nearest 16 byte cut.

After this, the remaining operations can be used on oracle queries. Our querying strategy is to loop over all 8 possible random paddings, and subtract those off of the data size we saw on the wire. Calling the length over the wire, and letting P_i be one of the 8 possible random padding values, then we look at the subtracted length v_i = L - P_i for each i in [1,8]. Then, using a table of article names -> true filesize, we look at all possible articles with filesizes in range [v_i - b, v_i]. In other words, we look at a small window of possible true filesizes for each i. For the sake of argument, we use b=15 to take care of the possible SSL/TLS padding. In practice, we could up this to a value like 50 to add extra wiggle room for other HTTP data not taken into account. However, such data is likely to be fixed size and hence can be subtracted off as well. Then, we will oracle query all such articles that have filesizes that fall into that range, for each i. We allow 240 oracle queries, which combined with our 59 active operations and single passive opertions satisfies the input budget of 300 operations.

To confirm that this strategy would work, a simulation script was written that implements the above strategy, which is included as "simulation.py". Additionally, all of the filesizes of the articles are needed, and included as "filesizes.txt". The simulation works with well over 95% accuracy.”

Live-Collaborative Evaluation:

Collaborating with others Two Six Labs identified the intended vulnerability

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.8 Poker

A.5.9.2.8.1 Question 004 (SC Space, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

“There is no side channel in space to determine the opponent's hole cards. The most reasonable approach was to exploit the fact that the server draws the current game table as a JPEG and sends that to both players, as the file sizes of the individual card images (in the "cardface" directory)

vary wildly. However, while this approach would have been effective, it is countered by the fact that random padding is added to the image before it is sent.

Specifically, sendTableToClient() in the PokerServer class makes a call to drawTable(), which generates the JPEG image. The JPEG image consists of overlays of cards, each of which is a PNG with varying sizes. This image is then base64 encoded, and sent to the players. Note that each player gets a different image, since the opponents cards are always replaced with "cardback" images. Normally, the image size of the JPEG being sent to the opponent is a strong enough side channel in space to determine their hole cards, however when sendMessageToClients() is called within sendTableToClient(), between 5000 and 95000 bytes of 'A' are added to the response before sending. The actual number of padding bytes appears to come from a good source of randomness. This thwarts any attempt at performing the side channel attack on the image size.

A crude python simulation was written to determine if the side channel is still possible with this random padding, given the operational budget. This simulation intentionally makes incorrect assumptions in favor of the attacker to ensure that the attack is not possible (for example, SSL/TLS is not considered, which makes the attack more difficult as sizes are binned into cipher block size multiples. Furthermore, we assume that the JPEG overlay process simply adds the filesizes of all overlays, which is not true). Even with these incorrect assumptions in our favor, the range of the random padding is too large to overcome for this attack. The files "simulation.py" and "filesizes.txt" are included for reference. The only other way in which a side channel in space can be utilized is the size of the strings of the card names that are sent to players. However, this was not considered heavily as the differences in lengths are at most a few bytes and would not be strong enough for a proper side channel, even ignoring the SSL/TLS and random padding."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No change.

Live-Collaborative Evaluation:

No change.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.8.2 Question 031 (SC Time, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"Yes, there is a side-channel vulnerability in time from which an attacker can discover the two hole cards dealt to the opponent. The side channel is a result of the timing differences in rendering the game table with different card images, which likely stems from the fact that the filesizes of the different card images have large variance.

To determine this, a modified version of the server was created. This version repeatedly made calls to sendTableToClient() for the opponent, and timed this function. The calls were made while looping over every possibly pair of hole cards the opponent could have. The timing data

for all possible hole cards for this function was output to a file. Additionally, each pair of possible hole cards was repeatedly used 75 times to obtain a mean and standard deviation for how long it takes for sendTableToClient() to complete. This data collection is all done offline, using no operations.

After this data was obtained, a simulation script was written. This script selects a pair of hole cards at random, and uses the observed mean and standard deviation for those hole cards to draw a sample timing from a normal distribution. Then, based on the data obtained, we query the hole cards whose mean timing is close to the observed sample. If the number of queries required to obtain the correct result is less than 700, this trial is a success. Using this strategy, we use 1 passive operation to observe timing, and the remaining operations as oracle queries.

This simulation results in 99% success within operational budget. However, this simulation does not take into account timing that is contributed from other sections of server code, as well as timing from network latency. We believe that timing contributed from other sections of server code is negligible to this technique, as the table rendering is the most time-intensive part of the server, and the other methods of the server appear to take an amount of time that does not vary anywhere near as much as the table rendering.

Regarding timing from network latency, it is believed that this can be normalized-out fairly well, especially since the current technique performs at 99%. Additionally, there are smarter techniques that can be incorporated that would result in even higher accuracy, which would further dampen any negative affect of network latency. For example, as of now, the process of performing oracle queries is by first querying the hole cards corresponding to the closest mean time from the obtained data, and then from there repeatedly moving up and down the sorted list of timing data. A smarter way to perform this search would be to simply keep querying the hole cards that correspond to the next closest mean time. Additionally, standard deviation can be taken into account to decide which hole cards to query next. Another piece of information to incorporate would be the timing of the first hole card dealt to the opponent, in addition to the timing of the complete pair that is dealt. This would likely drastically reduce the search space, as we could restrict our querying to pairs that include one card that makes sense based on its individual timing data. Lastly, we could perform more than 75 offline computations per card pair to obtain better data. The supporting script and data for this method is supplied, as "data.txt" and "simulator.py".”

Take-Home Evaluation:

The challenge program contains an intended vulnerability that leaks whether a card has been dealt under a new skin. There are small timing differences in the individual card independent of this, but this timing should dominate. Two Six Labs has reported a timing difference in the individual cards. It is unclear whether this would be sufficient under a lower budgeting scheme, however, as there exists an exploit to create an even segmentation of cards to use the intended side channel, the exploit described by Two Six Labs would provide a sufficient ordering where all the previously dealt cards would gravitate towards one mean and the previously undealt towards another. This is effectively indistinguishable from the intended side channel, perhaps better since it provides a method for ranking within the subsets segmented by the strong side channel.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.9 Searchable Blog

A.5.9.2.9.1 Question 018 (AC Space, Intended Not Vulnerable, Take-home: No)

Take-Home Response:

“No. The stochastic matrix used for the Page-Rank algorithm utilizes the majority of the memory used by the challenge program. Increasing the size of this matrix could potentially reach the target 2GB utilization. However, the user can only populate one new blog, which is given the hard-coded name "userblog.html". Because of this, the user can only increase the default size of the matrix by a single row and column, which is insufficient to reach the goal 2GB memory usage.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.9.2 Question 027 (AC Time, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“The challenge program runs the PageRank algorithm to rank blogs. The PageRank algorithm uses a stochastic matrix A to compute the page rank. It takes iterated products A, A^2, A^3, ..., A^n, ... until the result stabilizes. Normally page rank uses a damping factor to account for the fact that a user will eventually tire of clicking through on links, preventing the calculation of infinite walks. Typically, this factor is set as a=0.85, which degrades nicely. For the challenge problem, they set the damping factor to a=0.999 (see RankingService.performRanking()), which degrades very slowly.

When you upload a blog (the user can only upload one blog, given a default location of "blog/userblog.html"), the PageRank algorithm adds all of the outgoing links as entries into the matrix A. By submitting a blog that only links to itself, we force the lower right-hand corner of the matrix A^n to have value a^n=0.999^n. Since the products A^n are calculated until the result stabilizes, and 0.999^n degrades slowly, this maximizes the number of products the challenge program has to compute while updating the page rank for the new blog.”

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.10 SimpleVote 2

A.5.9.2.10.1 Question 008 (AC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“Yes. A successful request for all election results causes an excess amount of memory to be utilized. This is achieved by updating a voter's ballot 10 times. On the tenth update to the ballot, the system generates election updates for all valid elections, causing the challenge program to utilize over a GB of memory.

The exploit script assumes the attacker is "KathyRBown@rhyta.com", but the same attack would work with any valid voter user. The exploit script is run using the following syntax:

python exploit.py <target_ip> <target_port> KathyRBown@rhyta.com Iapeequu3dah

e.g. python exploit.py localhost 8443 KathyRBown@rhyta.com Iapeequu3dah

The script first logs in as the user KathyRBown@rhyta.com and registers a ballot for election 401 (of which Kathy has a valid registration key). The script then calls ten updates to this ballot. After each update is sent, there is a call to the EasySelectionServiceImpl.updateContest function which checks whether the number of contest updates is congruent to zero modulo 10. After the tenth update, this check passes, and the contest results are all generated. During this time, the challenge program reliably utilizes over 1 GB of total memory. Each election update request causes the LZW::zip function to be called, which requires hundreds of MB to be allocated for each call. Measured in Wireshark, the entire attack requires 9066 bytes, well under the input budget."

Take-Home Evaluation:

The challenge contains an intended vulnerability, but Two Six Labs appears to have discovered an unintended vulnerability. A feature added for an SC vulnerability (the ability of users to trigger synchronization) appears to result in the vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.10.2 Question 016 (SC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

"When a user attempts to log in, the LoginManager calls TokenedPasswordChecker.processPassword(), which returns a Report containing a boolean indicating whether or not the passwords match, and a token generated during processPassword(). If the passwords do not match, the redirect returned includes an 'id' parameter containing the result of LoginManager.grabSessionString(). The grabSessionString() function iterates through characters of the created Report token. Although pullTokenByte() completely masks the value of token characters by adding a random integer value between 0 and 9, and then returning the value mudulo 10, the length of the id attribute reveals the length of the token. The token created by processPassword() consists of the actual user password and the password entered by the user, separated by whitespace, and then compressed. By examining the output of processPassword() with different inputs, we identified that the token length after compression does not change if the entered password is a prefix of the actual password, whereas the length is larger if the entered password is not a prefix of the actual password.

Using this fact, we can use the length of the id attribute, which equals the length of the token, as an oracle for whether our password guess matches a target user's password. We first submit an empty password field. The empty string is always a prefix of the actual password, so this reveals the target token length. We then guess all single character passwords until the id attribute matches our target value, at which point we have learned the first character of the password. We then proceed to the second character, and so on, until we have determined the entire password. Once we have determined the entire password, we will be able to successful log in as the target user and access the target voter's profile.

The constructor for TokenedPasswordChecker requires secrets to be between 5 and 15 characters long, and contain only ascii characters between '!' and '~' (i.e. all 10 digits, 52 upper and lowercase letters, and 32 special characters). In total there are 94 possible characters that could be used in a password. Our script iterates through all possible characters, checking letters, then

digits, then punctuation. The last character checked is '~', so with the maximum password length of 15 characters the worst-case password is '~~~~~~~~~~~~~~~'. This password would take 94*15=1410 active operations to determine the full password, and then an additional two operations to login and access the profile page, for a total of 1412 operations, which is within the 1500 operation budget."

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.11 STAC SQL

A.5.9.2.11.1 Question 013 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

"No, there is not algorithmic complexity vulnerability in space that causes the challenge program to store a file with a logical size exceeding the resource usage limit given the input budget. There *is* an algorithmic complexity vulnerability in the program's use of Huffman Trees for compression of `blob` data. However, this vulnerability cannot be exploited to produce a file of greater than 2 MB within the input budget. As detailed below, the vulnerability is exploited by creating a table with one or more columns of `blob` data, inserting several rows with value `AA` (base64 of `0x0`), repeatedly updating all rows with new values, and then setting all rows and columns to a large number of repetitions of `A` (ie a large number of zero bytes). There are thus four variables: the number of columns in the table, the number of rows inserted, the number of updates performed, and the length of the final string of `AAAAAA...` sent in the final update. Optimizing over these variables within the constraints of at least 2 MB of output in `db.stacsql` and less than 12 KB of PDU usage requires that the final update send approximately 3000 bytes of `0x0` — ~4000 bytes of `A...`. This, however, is impossible, as `SQLInputWorker.run()` truncates commands from the client to 1000 characters. Adding this as a constraint, the optimum solution (under conservative approximations of the PDU cost of additional rows and columns, etc) now requires 3 columns and approximately 60-65 rows and updates, with a final update of the maximum of just under 1000 bytes to each column. However, this results in only just over a megabyte of data in `db.stacsql`, and so we say that the vulnerability is not exploitable within the PDU limit.

(The only other file writes done by the challenge program are done to the log file, but there is no amplification of user-submitted data in the logging [ie n bytes of user input does not cause the program to write significantly more than n bytes to its log file], and so the logging subsystem is not an attack vector.) (Note that under a more generous interpretation of the PDU limit the attack listed in the accompanying `exploit.txt` files causes the program to write a file of logical size approximately 2.5 MB, over the resource usage limit of 2 MB.)

The vulnerability lies in the program's use of Huffman coding to compress field values. When a new row is created a Huffman tree is created for each field in the row. This tree, created via `HuffamnCoder.constructTree()`, contains entries for only those byte values present in the field's values. When the value in a field is updated, eg by a `update` statement, the function `SQLField.setData()` is called. If the field is of type `BLOB` and the field already has a Huffman tree (ie already has a value), the `HuffmanCoder.updateTree()` function is called. This function takes the existing Huffman tree for the field, makes a new tree for any byte values in the new

field value that are not in the existing tree, and then joins the old and new trees into a single new tree by setting them as left and right children, respectively, of a new parent node. Thus, every time a `BLOB` field is updated with a value containing byte values not previously present in that field, the Huffman tree for that field gets a level deeper, as the existing tree is pushed down a level by the creation of a new parent node, and so the Huffman codes for any byte values already in the tree get a bit longer.

The attack thus proceeds as follows: create a table with a single column of type `BLOB`, then add 256 rows to the table. Each row contains a single byte of `0x00` (`AA` in base64). Issue a sufficiently large number of `update` statements where each one sets all rows to a byte value not previously seen, thus pushing the node for byte value `0x00` further down the Huffman tree and lengthening its code via the mechanism described in the previous paragraph. Finally, issue an `update` statement that sets every row to a very long string of `0x00` bytes (`AAAAAAAAAA...` in base64), thereby forcing the program to store the very long Huffman code for `0x00` once for each zero byte in each of the 256 rows. A file size of 2483738 bytes is thus achieved."

Take-Home Evaluation:

The challenge program contains an intended vulnerability. The Huffman compression scheme can be used to exceed the resource usage limit. Two Six Labs appears to have identified the intended vulnerability but reported the challenge not vulnerable on the condition that the attack does not satisfy the input budget. The challenge has a very narrow budget to prevent unintended vulnerabilities.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Yes, there is an algorithmic complexity vulnerability in space that causes the challenge program to store a file with a logical size exceeding the resource usage limit given the input budget. We have changed our answer based on collaboration with GrammaTech.

For a proof of concept, run the server via the `startServer.sh` script included with the program. Run `exploit.sh`, passing the host of the server. After executing the commands in this file the main file used by the program, `db.stacsql`, will have a logical size of 2483738 bytes as reported by `stat`. The exploit script uses ~10 KB of PDU, within the input budget of 12 KB. […]"

Live-Collaborative Evaluation:

Two Six Labs identified the intended vulnerability during the take-home but responded "No" as the exploit used a client that prevented the team from staying within the input budget limit. By recognizing that the attacker is not constrained by a client, they used another client to fit the exploit within the input budget.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.12 Stegosaurus

*A.5.9.2.12.1 Question 003 (SC Time, Intended Vulnerable, Take-home: *Yes, Live: *Yes)*

Take-Home Response:

"No, there is no side channel in time that reveals the user's secret message. As the attacker all we can assume is that we will witness the target user stegging one image one-time with one

message. As the stegged image, key, and secret message are not stored in memory or on disk after the stegging, any active operations performed after the target user's stegging will not leak the needed information. The only endpoint that leaks information about the user's message is progress.data. As the stegging process is occurring, there is a "status" variable that gets set to the position of the current message bit that is being stegged. Querying the progress.data endpoint reveals which bit of the message is currently being stegged, and timing this can perhaps reveal information about the message. However, due to the 50 millisecond sleep statements in HeartbeatThread.getStatus() and HeartbeatThread.run(), queries to progress.data do not provide very granular information. Furthermore, the timing information is not solely a function of the message, but of both the message and the key, and we do not have a means of obtaining the key, as SSL/TLS is used to encrypt packets. (The key and message are mixed in the Stegger.hide() method, and this mixed timing is what would be observed.)

We did however identify a way of recovering the secret key without leveraging the timing of network packets. The attack works as follows:

1) Before the target user's request, send a request to hide a secret message. This will reveal the current id in use by the system.

2) We assume the target user is the only other user interacting with the system, and ids are handed out sequentially, so we can determine what their id will be:

a. if the user starts their encoding during ours, the id will be one more than our id

b. if our request returns before the user starts their request, the id will be the same as our id, because the id is reused once a request is complete

3) Watch the network for the target user to begin encding their secret message

4) While the target user's message is being encoded, run `curl <stegosaurusHost>:8080/done.html?id=<ID>` replacing <ID> with the id from step 2

5) The response to the curl request will be html that includes the secret message in plaintext

We did not consider this a side channel under the operational definition due to the fact that the attack does not rely on timing information between pairs of network packets. The attack requires only knowledge of the presence of network packets, which signal the user is performing their encoding. The attacker does not require timing information to identify the start of the user encoding, as no other users are interacting with the system. The attacker does not need to worry about missing the window for attack, since the user's encoding will take at least 5 seconds, due to the delay between polling for progress and requesting the final stegged image. Finally, the operational budget allows for up to 5000 active requests, so the attacker can begin sending curl requests as soon as the target user begins interacting with the server."

Take-Home Evaluation:

The out-of-scope leak identified by Two Six Labs was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"[…] In our live engagement collaborations, it was observed by a number of teams that the size of the image, key, and message all play a role in the timing. Teams could identify differences for certain values of the key and image that leaked information about the message, however without

control over the image, key, or message, teams agreed that there were worst case values for which the message bits could not be recovered. Additionally, we note that the 5000 operation budget precludes sending many requests in an effort to get more granular data from the progress endpoint is not feasible since the message that is recovered may have as many as 512*8 = 4096 bits. […]”

Live-Collaborative Evaluation:

The out-of-scope leak identified by Two Six Labs was confirmed.

Post-Live-Collaborative Analysis Ruling: Correct

A.5.9.2.13 StuffTracker

A.5.9.2.13.1 Question 030 (AC Space, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“In the SDL schema, an inventory report upload consists of a single set of list tags enclosing one or multiple item tags. However, the ItemListHandler, which is used while parsing an inventory report, does not implement this schema exactly. The ItemListHandler.startElement() function ignores all tags except <item> tags, ignoring all <list> tags. The ItemListHandler.endElement() function processes every </list> tag and calls createSDLDoc_and_LocalDictionary(). The end result is the items in the inventory report are stored in the index n*(n+1)/2 times, where n is the number of nested <list> tags in the inventory report. We noted that uploading a document with nested <list> tags did not raise any errors in the server as long as there were at least as many items inside the lists as there were <list> tags, and all the items contained a 'c' attribute and had the same 'id' attribute. We note that this upload process is vulnerable to an AC Time attack, as providing more nested <list> tags can cause the upload request to take arbitrarily long.

When issuing an inventory search request, each index entry is processed by the IndexDecompressionHandler, which adds items matching the search term into the resultset. The resultset is stored in memory in a StringBuffer, so adding many results into the result set can drive the JVM memory usage very high. Uploading a document with n nested <list> tags containing m identical items, and searching for a term that matches that item causes m items to be added to the result set for each of the n*(n+1)/2 entries in the index, for a total of m*n*(n+1)/2 entries in the result set.

We ran our exploit script using 1000 items within 100 nested lists. The sum of the PDU sizes of the application requests was 28548 bytes. The script finished in approximately 7 minutes and resulted in a total memory usage of 5682096 kB as reported by pmap.”

Take-Home Evaluation:

The challenge contains an intended vulnerability. Two Six Labs appears to have discovered an unintended vulnerability that interacts with the same components as the vulnerable code.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.14 Tawa FS

A.5.9.2.14.1 Question 012 (AC Time, Intended Vulnerable, Take-home: Yes)

Take-Home Response:

“The challenge program implements a custom CRC hash function. This function contains a subroutine called "find_high_bit" which inefficiently scans from right to left testing for whether the hashed array contains a set "1"bit. The CRC hash is only called by the BaseBlock calculateHash() function, which always performs a hash of a 4095 byte new array (zeroed out). Each time the do_interface function is called, the challenge program computes three CRC hashes for each file in the indicated directory. By filling a directory with as many files as possible and optimizing the number of calls to the "interface" page, we can achieve the desired resource usage. During that time, a benign user cannot communicate to the challenge program, making the benign user request to exceed the allotted time. For the input budget of 40000 bytes, the attacker has only access to 38673 for the attack (718 bytes are required for registration and609 are required for logging in initially). Each file upload takes 564bytes. Each GET request for the interface page takes 241 bytes. If A is the number of files, and B is the number of interface requests, then we have564*A+241*B <= 38673 and also the total number of CRC calls is equal to A*B*3Solving the first equation for B yields B=(38673-564A)/241. Substituting this into the second equation and optimizing finds an optimal value of A~34, which yields B~80. These were the allocations that we chose in our exploit script. The provided exploit script starts by registering and logging in as a new user (with username and password each a character). It then creates and uploads 34 files to the challenge program into the home directory of the malicious user. Then 80 GET requests are sent to the challenge program for the interface page. Once all of these requests are sent, the benign request is sent and timed. The timing only covers the individual benign request and does not take any of the other request timings into account.”

Take-Home Evaluation:

The challenge contains an intended vulnerability. It appears that Two Six Labs identified an unintended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.5.9.2.14.2 Question 020 (AC Space, Intended Vulnerable, Take-home: No, Live: Yes)

Take-Home Response:

“Each individual user is given a file that holds their file-system, givenby <username>.fs in the /tmp directory. These files are initialized tobe 104.8576 MB, and we found no way to alter the size of that file withinthe input budget. By signing up several users, one could create several of these files and thus utilize 100+ GB of combined disk space within theinput budget, but the question specifically asks for the program to storea *single* file, so this method is insufficient.”

Take-Home Evaluation:

The challenge contains a race condition that can be used to trigger a non-terminating loop using disk space on each iteration.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“There is a race condition involved in the defragmentation process. Essentially when the server performs defragmentation it copies the contents of a 4096 byte block to a new location, makes a check to see if the new block and previous block are equivalent, and continues if they are not equivalent. By submitting many defragment requests on a fragmented file, we can cause the challenge program to alter the block while it is in the process of copying it, causing it to enter an

infinite loop. While the user's .fs filesystem size climbs slowly, it will eventually eclipse the target size (and beyond).”

Live-Collaborative Evaluation:

Two Six Labs identified the intended vulnerability after collaboration.

Post-Live-Collaborative Analysis Ruling: Correct

A.6 Engagement 6

A.6.1 University of Colorado Boulder

A.6.1.1 UC Boulder Overview

Colorado answered 26 challenge questions with an accuracy of 73% in the take-home and 100% in the live-collaborative engagement. Colorado answered 3 more questions in the Engagement 6 take-home than in the Engagement 5 take-home and lost 1 percentage-point in accuracy. Iowa, GrammaTech, Colorado, and Vanderbilt all achieved between 73% and 76% accuracy in the take-home. The team answered the fewest number of questions and tied for the fourth highest accuracy in the take-home engagement. Against the segmentation of all questions (treating un-answered questions as incorrect), the team had the lowest accuracy of 44% during the take-home engagement.

Colorado achieved the highest TPR (75%) against the segmentation of answered questions, tied with the control team; however, the team had the second lowest TNR (72%). Accounting for the heavy bias towards non-vulnerable challenge questions in this engagement, the TPR and TNR values provide a clearer view of the strengths and weaknesses of the team’s capabilities than the accuracy. The strength of Colorado’s approach is in identifying vulnerabilities, but the team has difficulty ruling out vulnerabilities relative to other performers. Against the segmentation of all questions, due to answering fewer questions than all others, the team had the lowest TNR (36%), but achieved the second highest TPR of 60%.

During the take-home engagement, Colorado had the most difficulty with SC Space questions (50% accuracy) and SC Time/Space questions (67% accuracy) and achieved above a 70% accuracy in all other categories. In our assessment of reasons for incorrect side channel responses, failure to correctly analyze strength was the dominant reason. In the case of incorrect algorithmic complexity responses, the reasons were evenly distributed between not identifying the complexity, not understanding the inputs to the complexity control flow, and incorrectly determining the complexity bounds.

During the live-collaborative engagement, all R&D Teams achieved between a 97% and 100% accuracy, with all teams achieving a 100% TNR in the segmentation of answered questions. Collectively, the R&D Teams identified 3 unintended vulnerabilities during the live collaborative engagement – Railyard Question 013, Class Scheduler Question 017, and SimpleVote Question 038. Railyard Question 013 and Class Scheduler Question 017 came out of a collaborative effort from all teams. The SimpleVote Question 038 unintended vulnerability was confirmed with a clever exploit by Northeastern.

A.6.1.1.1 Take-Home Engagement Scores

Table A-117: Engagement 6 UC Boulder Take Home Engagement Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	4	2	50
SC Time	4	4	100
SC Space/ Time	3	2	67
AC in Space	7	5	71
AC in Time	8	6	75
Total	**26**	**19**	**73**

Table A-118: Engagement 6 UC Boulder Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	12	9	75
Not Vulnerable	14	10	71
Yes Answer	13	9	69
No Answer	13	10	77

A.6.1.1.2 Live-Collaborative Engagement Scores

Table A-119: Engagement 6 UC Boulder Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	4	4	100
SC Time	4	4	100
SC Space/ Time	3	3	100
AC in Space	7	7	100
AC in Time	8	8	100
Total	**26**	**26**	**100**

Table A-120: Engagement 6 UC Boulder Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	12	12	100
Not Vulnerable	14	14	100
Yes Answer	13	13	100
No Answer	13	13	100

A.6.1.2 UC Boulder Specific Responses

A.6.1.2.1 BattleBoats 3

A.6.1.2.1.1 Question 022 (AC Space, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Yes, there exists an algorithmic vulnerability in space that could cause the program's memory usage to exceed 1000MB. In Battleboats2 (from Engagement 5) the attacker could trigger an AC vulnerability in space in the method ccsc.HitLocator.computeHitTime, which computed the travel time of the ball shot from the cannon. In Battleboats3, the procedure ccsb.HitPinpointer.determineHitTime computes the travel time of the ball shot from the cannon in a very similar manner: This method tries to compute the time in a loop until it cannot find a better estimation of the travel time with the fixed precision of 1 E -100. In order to find the desired time, the while loop keeps putting the calculated times (which are computed in the method sscb.HitPinpointer.step) with their corresponding height to the HashMap<BigDecimal, BigDecimal> trajectory until either it cannot update the time with the given precision, or it tries to insert an value which already exists in the map. The step method computes the time up to 400 iterations, and after that it always returns the same value. Therefore, the while loop in determineHitTime method is executed at most for 400 iterations. This 400 iterations limit, which is checked at the beginning of the procedure ccsb.HitPinpointer.step, ensures that the trajectory HashMap never inflates the memory. However, another algorithmic vulnerability in space exits in Battleboats3.

Despite the HitLocator.advance in Battleboats2 which used the TrajectoryData from round n - 1 to find the data in round n, the HitPinpointer.step procedure recalculates the data generated in round n-1 in round n. This means in each iteration of the while loop in method HitPinpointer.determineHitTime, 26 BigDecimal operations are executed (including 16 multiplications). These operations allocate additional memory, and therefore result in a potential space vulnerability. In order to trigger this vulnerability, we can send shoot commands that cause the while to be executed for 400 iterations, and depending on how the behavior of the garbage collector, we can see the memory usage exceeding the 1000 MB limit. We have included an exploit which triggers this vulnerability in the directory exploits/battleboats3/exploit22."

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Iowa state further investigated our exploit presented below and narrowed down the active part of the exploit to the velocity and argued convincingly that the explanation for the exploit is in the method HitPinPointer.sqrt where a BigDecimal with a large number of digits is converted to a double, then squared and subsequently converted back to a BigDecimal. This value, being an intermediate result, needs to be kept in memory throughout the computations, causing the large increase in memory observed.

Live-Collaborative Evaluation:

Colorado collaborated with Iowa to gain a better understanding of their proposed exploit.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.1.2 Question 043 (SC Time, Intended Not Vulnerable, Take-home: No, Live: No)

Take-Home Response:

"No, there is no side channel vulnerability in time that allows the attacker to determine the position of their cannon on the opponents' board. This question is similar to the question 10 in Battleboats2 (from Engagement 5), where the attacker could take advantage of a timing side channel that can be observed between the attacker issuing a shoot command and the victim issuing a response. Despite Battleboats2, in this version the attacker will observe a similar response time regardless of where the shot goes on their radar board (i.e. if they have fired outside or inside of the games ocean board). Similar to the method Competition.isOver in Battleboats2 which caused the vulnerability, the ccsb.Tournament.isOver procedure in Battleboats3 has an unconditional waiting on an object for 30 ms. The CodeHawk cost model below shows the cost report prodcued by CodeHawk cost model, and as you can see there is a very high constant cost for the method Tournament.isOver. Inspection of the reverse call graph for this method, shown in Figure 2, shows that the method is called in the handling of a hit result message. However, in contrast to Battleboats2, calling the method Tournament.isOver is included in all paths from the caller method BattleBoats.fixHitResults (see Figure 1. Since the procedure BattleBoats.fixHitResults is called for any shot which hits a location in the radar, there is no difference in the response time of shots which are within the board boundaries and the ones which are outside of the board. Therefore, the attacker cannot observe any difference in the response of the shot response as long as the shot hits a location on the radar."

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"After discussions with Iowa State we strengthen our justification that no side channel in time exists. Starting from the method shotTaken, the descendant call graph shown in Figure 1 we enumerate the potential methods that could potentially participate in a side channel. The figure shows that two methods called have an undetermined cost, HinPinPointer.init and HitPinPointer.determineHitSetting. Looking at the cost model results for HitPinPointer.determineHitSetting, we see a constant cost plus the cost of two callees, one of which again has a fixed cost plus the cost of two other methods, for which we also add the callees: […] All of these methods and their callees have either a fixed cost or small differences which cannot contribute to a side channel. The only potentially interesting method is OceanScreen.pullHits, whose control flow graph, shown in Figure 2, exhibits a pattern that could support a side channel. The control flow graph, annotated with branch conditions and costs, where an X indicates a symbolic expression, shows several paths controlled by branch conditions related to the position of the hit in the square, where some of the paths are very cheap (5 ns) or have a symbolic expression that depends on the callees of the respective basic block. Looking at the cost model for OceanScreen.pullHits […] we see that all contributions to the cost come from calls to stream functions that iterate over some collection. Stepping through these functions together with Iowa State, using their tool, we jointly established that none of these calls is sufficiently expensive to support a side channel, which is in accordance with out observations

that the time differences are too small to extract the required information. In a subsequent session Iowa State tried to reproduce the exploit claimed by Northeastern, but was not successful, so the claimed exploit was not considered convincing."

Live-Collaborative Evaluation:

Colorado collaborated with Iowa to gain confidence in their take-home "No" response.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.2 BraidIt 3

A.6.1.2.2.1 Question 027 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"No, there is no space side channel through which the attacker can get the index of the braid selected by the victim. Let the size of five unmodified braids be b1; b2; b3; b4 and b5, the size of modified braid be n, and the total packet size observed for a round be P. The attacker knows the actual length of the modified braid n and also the lengths of five unmodified braids b1; b2; b3; b4 and b5, in addition to the received packet size P. We use our functional extension of Discriminer to debug functional non-interference. The usual non-interference property is defined as the following for space side-channel: There is a space side channel (violation of the property), if for two selected braid indices (secrets), the size of received packet P (observation) for the same values of five unmodified braids and the modified braid (public inputs) differ from each other. Since the size of unmodified braids and the modified braid
is not deterministic (randomly chosen), we consider the functional non-interference property as following: There is a space side channel (violation of the property), if for two selected braid indices (secrets), the functions (one for each selected braid) of the received packet size P as the model of unmodified braids and the modified braid sizes $n + b1 + b2 + b3 + b4 + b5$ (public inputs) differ from each other. In other words, there is no functional space side channel, if we consider the size of received packet as the function of size of five unmodified braids and the modified braid and for any selected secret braid, we observe the same function. Next, we debug functional non-interference for two selected braids. We design this experiment for two secret braid selections (1 and 2) and obtain corresponding functions. Figure (3) shows two packet size function P versus the total size $n + b1 + b2 + b3 + b4 + b5$ (sum of the modified braid length and the lengths of the five unmodified braids) for selected braid indices 1 and 2. As a result, we observe the same packet size functions for both selected secret braids. Our functional clustering algorithm also returns 1 cluster for a given small indistinguishably distance. This observation is true for any two secret selections in both rounds 1 and 3. In the source code, we can identify the reason for the non-existence of the space side channel: in braidit3, the method phase.PlaitSelectedPhase.takePlaitString(), which was the cause of vulnerability in the other versions of braidit, is not vulnerable because the selected secret braid (say i) is not correlated with the received packet length. In other words, for any selected braid index, $P = n + b1 + b2 + b3 + b4 + b5$."

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.2.2 Question 029 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"No. There is a vulnerability in RSA decryption procedure, but it is not exploitable under the given budget. Braidit 3 uses the Chinese Remainder Theorem. Each player will challenge another one to make sure that he is really in possession of the secret key associated to the transmitted public key. To do this, a pair A sends another pair B a random value, encrypted with his public key, and asks B to decrypt it and send back the value as the proof. We apply Discriminer to see if we can detect the side channel in the connection step of the game. First, we consider RSA encryption part of the application and challenges the decryption part of it. We consider 4 pairs of secret keys and we collect 4,125 different guesses that should be decrypted using the secret keys. The inputs are generated with the domain knowledge and American fuzzy lop (a security-oriented fuzzer that employs a novel type of compile-time instrumentation and genetic algorithms to automatically discover clean, interesting test cases that trigger new internal states in the targeted binary). We run the application two times:

1) The original version of the application to measure the execution time.
2) The instrumented version of the application that records the function calls and basic blocks.

Second, in the execution time domain, we apply clustering algorithm to discover different classes of observations. Figure (4) shows the results of the clustering step. As there are three different clusters, we want to explore the root cause of the timing differences based on properties of program internals. Third, we combine the features of program internals alongside the cluster labels for each execution trace and apply decision tree learning algorithm. In this case, the decision tree algorithm shows that the number of times the program takes the basic block corresponding to the following conditions explain the differences between cyan and green clusters:

$$if\ (u.compareTo(this.M)\ >\ 0)\ \{$$
$$u\ =\ u.mod(this.M);\}$$

Equation A-9: BraitIt Challenge Cluster Analysis Equation

Note that, in the level of basic blocks, there are no features that can explain the differences between the red and green clusters. So, we conclude that the differences between green and red clusters are due to the presence of noises. Looking into source code, we can see that when the guess is a multiplier of the secret, the condition in the montgomeryMultiply of MontgomeryTimeser class is taking less time than otherwise. Using this vulnerability, one can

narrow an interval around a multiple of the secret prime to the point that it is trivial to find it by performing a small number of gcd computations against the public modulus. The most known attack for us (from Engagement 4's proof) to recover the secret key requires more than 60,000 operations. So, we are not able to obtain the secret key with 40,000 operations."
Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.3 BraidIt 4

A.6.1.2.3.1 Question 032 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)
Take-Home Response:

"Yes, there exists a side channel in space from which an attacker can determine the braid selected by the victim in both round 1 and round 3. The vulnerable method is com.cyberpointllc.stac.phase.state.WeaveChosenState.obtainWeaveString(). In this method, the modified braid (say of size n) is padded to size $i * n$ where i is the selected braid index. Then the five unmodified braids (say of length b1; b2; b3; b4 and b5) are appended to the padded modified braid and this packet of length $P = (i * n) + b1 + b2 + b3 + b4 + b5$ is sent to the attacker. Since the attacker can obtain the lengths of the five unmodified braids $b1; b2; b3; b4; b5$ and also the modified braid without the padding n (the application will display this information to the attacker), it is easy for the attacker to determine the selected braid index i from the received packet length P. We apply functional version of our tool Discriminer where we model the observation (total received packet length P) as the function of public inputs $(n + b1 + b2 + b3 + b4 + b5)$. Then, we apply functional clustering algorithm to discover different classes of observations. shows the plot of the received total packet length P on the y-axis (observations) versus the sum of the modified braid and the five unmodified braids n+b1+b2+b3+b4+b5 (public inputs) for the braid selections of 1 and 2 (secrets) shown with green and red series, respectively. In this case, our functional clustering algorithm finds two clusters. More specifically, for two selected secret braids, there are two different total packet size functions in the domain of the total size of the modified braid and the five unmodified braids. In other words, the variation in the observation (total packet size) is not explainable with public inputs (size of the modified braid and the five unmodified braids) and the selected secret braid is correlated with the observations. Number of operations needed: The attacker sends the connection request (one active operation), offers the game with some number of strands in round 1 and round 3 (2 active operations) and observes the received packet length in both rounds (2 passive operations). Hence the side channel attack requires 5 operations in total."
Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. The index orders are shuffled and there are not enough oracle queries to map the leaked unshuffled index to the shuffled indexes received by the attacker.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After discussions with GrammaTech, Iowa, Vanderbilt, we decide to not change our answer, but we improve our justifications. Since braidit randomly shuffles the braids, the index of selected braid is not the same as the one chosen by the victim. The only way that the attacker can guess the selected braid is to undo the shuffle procedure. We are convinced by GrammaTech that there is a possibility to undo the shuffle procedure with JAVACG1where given the braids received by the attacker, it predicts the first few characters for every positions of the original braids. The vulnerability is in grabRandomWeave method that makes the random generations of the braids to be predictable. Note that since both nextInt and nextBoolean uses the same state of random generations, the attacker sets the numStrands to 26 to have a higher chance of the successes."

Live-Collaborative Evaluation:

We believe that GrammaTech's exploit to undo the shuffle procedure is feasible within the operational budget. However, as the java random library is out-of-scope, we have noted this as an out-of-scope vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.4 Calculator 1

A.6.1.2.4.1 Question 014 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Yes, there is an algorithmic complexity vulnerability in space. The exploit is under the directory E6-solutions/calculator 1/exploit 014. The vulnerable code is in the recursive method stac.cruncher.Transform.primaryTransform. Upon every invocation to method primaryTransform, the array paddedArray (which is constructed from a method argument) is split and then copy-pasted into 32 new sub-arrays, where each sub-array is passed as an argument to a recursive invocation of method primaryTransform. Therefore, there will be 32 more invocations to method primaryTransform upon every invocation to itself. The array duplication as well as the recursive invocation may lead to considerable memory usage. To exploit the vulnerability, we need to have method primaryTransform perform computation on numbers that have lots of digits. Since method primaryTransform is invoked when performing multiplication, we need to:

1. Generate a large number that has lots of digits.
2. Perform multiplication on the large number.

To generate a number that has as most digits as possible while consuming as least input budget as possible, we could do exponential computation. The exponential computation is achieved with multiplication using the equivalence $x^n \equiv x * (x * x)^{\frac{n-1}{2}}$. Meanwhile, there is an input sanitization before each time performing exponential computation: $X * Y \leq 5000$, where X is the number of digits of the base (which refers to arbitrary intermediate operand – the value of

($x * x$) in this case) and Y is the value of the exponent. Hence, one of the large numbers that we can generate with a small input size is 999^{1695} (8 characters in total). Then we perform multiplication on 999^{1695}, which is $999^{1695} * 999^{1695} * 999^{1695}$. To stay under the input budget, we send 60 requests where we ask the program to evaluate expression $999^{1695} * 999^{1695} * 999^{1695}$ in each request. […]"

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. Colorado similar to other teams identified an unintended vulnerability in the exponentiation calculation similar to other teams.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Yes, there exist another exploit from Iowa and a new vulnerability from Northeastern. Exploit by Iowa: Create a 50000-digit number by performing multiplication among several numbers who are powers of 9. Vulnerability: We believe the underlying vulnerability is the same as the one that we mention below. Exploit by Northeastern: Perform division by 1 repetitively on a very large number (which is the dividend). Vulnerability: The division method allocates new arrays upon each invocation and the objects and these are being appended in the method invoke to the final result, creating a never larger object that stays in memory during the entire computation."

Live-Collaborative Evaluation:

Collaborating with Iowa and Northeastern, Colorado identified additional unintended vulnerabilities.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.4.2 Question 016 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Yes, there is an algorithmic complexity vulnerability in time. The exploit is under the directory E6-solutions/calculator 1/exploit 016. The vulnerable code is in method processOutcome in file stac.cruncher.Cruncher. Method processOutcome uses an operand stack to store temporary computation results. However, if we feed as input a sequence of bracketed expressions, it will evaluate all bracketed expressions, but only return the value of the last bracket expression as the result. For example, if we feed as input $(1 + 2)(3 * 4)(2^5)$, it will first evaluate expression $(1 + 2)$ but then will immediately discard the result before evaluating expression $(3 * 4)$. Similarly, it will discard the value of $(3 * 4)$ before evaluating expression 2^5. The final result will be 32. This is vulnerable because although there is a constraint on the number of digits of every operand (i.e. 50000 digits) and therefore it is not possible to increase computation time

by performing multiplication on two infinitely large numbers, it is possible to increase computation time instead by wasting lots of (expensive) computation – which is exactly those

computation on bracketed expressions whose results are discarded immediately after evaluation.

While exploiting the vulnerability, we discovered that there is a limit on the number of characters in one HTTP request, which is 10240. Therefore, we need to find a reasonably large size

input. We choose as input: $\overbrace{A \ldots A}^{120}$, where we let A be $(999^{1666} * 999^{1666} * 999^{1666} * 999^{1666}) * (999^{1666} * 999^{1666} * 999^{1666} * 999^{1666} * 999^{1666})$."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. Colorado identified an intended vulnerability that was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.5 Calculator 2

A.6.1.2.5.1 Question 003 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"No, there is no algorithmic complexity vulnerability in space. Using CodeHawk we identify possible places where data may be written to file. To obtain the locations where values are written to file we identify all calls to the logging functions: info, warn, debug, and error. Only two calls are reported for warn […]. No calls are reported for the other logging functions.

The code in file stac.webcontroller.coach.AbstractHttpCoach is

```
1. catch (InterruptedException e) {
   AbstractHttpCoach.logger.warn("Unexpected interrupt received
   while processing HttpExchange", e);}
2. catch (ExecutionException e2) {
   if (e2.getCause() instanceof IOException)throw
   (IOException)e2.getCause();
   AbstractHttpCoach.logger.warn("Unexpected execution error
   received while processing HttpExchange", e2);
   }
```

There are in total four types of exception that could be thrown by the calculators: InvalidEquationBad, ExpensiveOperationBad, ReportTooBigBad and IllegalArgumentException. However, they are all subclasses of RuntimeException, which are neither subclass of InterruptedException nor of ExecutionException. Therefore, the path constraints are not satisfiable to reach the above two lines that write into the log file, making it not possible to increase the size of the log file. Using the same script we find a number of locations where data is written to a writer or output Stream […] The first call represents the response to the user, which is either returned to the web interface or printed to console (which we consider to be not a file). The second call (in fetchTemplate) writes to a StringWriter, which is discarded. The remaining calls (in the methods starting with PART) can be shown to be not reachable."

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"No, there is no algorithmic complexity vulnerability in space. Although GrammaTech manages to create a log file of 76KB by sending an expression surrounded by 9850 nested pairs of parentheses, it is beyond the input budget to reach a 4MB log file."

Live-Collaborative Evaluation:

Colorado correctly concluded that GrammaTech's potential exploit is incapable within the input budget.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.5.2 Question 005 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

Same as Question 016

Take-Home Evaluation:

The challenge program contains an intended vulnerability in processing operators in parsed expressions. Colorado identified an unintended vulnerability that was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Yes, there exists another vulnerability as mentioned by Iowa and Northeastern: The server will hang for some time when the input expression contains a multiplication between two very large numbers."

Live-Collaborative Evaluation:

Collaborating with Northeastern and Iowa, Colorado reported another unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.5.3 Question 009 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"No, there is no algorithmic complexity vulnerability in space. The recursive method chiefReformulate in file stac.mather.Reformulate only splits and copy-pastes the array paddedValues into 4 new sub-arrays (instead of 32 as in calculator 1), which consumes much less memory. We tried the exploits for Question 016 and Question 005 here, but they are both not sufficient to trigger the program to reach a memory usage of 4 GB. All other methods consume memory of linear size in terms of the input size."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability; however Northeastern and Two Six Labs identified an unintended vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Yes, there exists two algorithmic complexity vulnerabilities in space. Northeastern believes there is a vulnerability same as their answer to Question 14: Repetitive division by 1 on a very large number will lead to excessive memory usage, since the division method is the same here as in Question 14. Iowa believes there is a vulnerability same as their answer to Question 5: The multiplication between two very large numbers will lead to excessive memory usage. The exploit is composed of a multiplication between a 48000-digit number (every digit is ‘9’) and a 1001-digit number (every digit is ‘9’).”

Live-Collaborative Evaluation:

The unintended vulnerabilities discovered by Northeastern and Iowa were confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.6 Chessmaster

A.6.1.2.6.1 Question 006 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“No, there is no side channel in time through which an attacker (with knowledge of the target user's username) can obtain information that enables him to determine the current state of the target user's game under the given operational budget. The only way the attacker can determine the current state of the target user's game is by obtaining the password of the target user via the side channel and then simply giving a display request to the server from the target user's username and password. The server will then reveal the current state of the target user's game to the attacker. It remains to establish whether it is possible for the attacker to obtain the target user's password from a timing side channel. When any request is sent to the server, it checks whether the username and password is valid in the com.ainfosec.chessmaster.Server.CheckLogin(). After checking whether the provided username is in the database, this method then uses Java's inbuilt String.equals() method to check whether the provided password string is equal to the password associated with that particular user. String.equals() works by initially checking whether the two strings are of the same length and then comparing both strings character-by-character and returning a false value as soon as the first character mismatch is detected. An attacker with knowledge of the target user's username can exploit this behavior to his advantage in the following way:

- Initially the attacker provides 11 requests with random passwords of lengths ranging from 10 to 20. The String.equals() method will take slightly longer to return false for the active request for which the password length is correct.
- Once the password length is known (say n), the attacker determines the first character of the password by sending k requests (where k is the number of allowable characters) of length n, each with a password whose first character is one of the k allowable characters. The method will take slightly longer to return false for the request that has the correct first character.
- The attacker continues this way for each of the remaining $n - 1$ characters. In the end, he has the full password of the target user, with which he can give the display command to the server to obtain the current state of the game being played by the target user.

In the given application, there are 43 allowable characters in the password. If the timing difference between the cases when first k characters of the password are correct and the first $k +$

1 characters are correct is distinguishable, it would take a maximum of $11 + 43 * 20 + 1 = 872$ active operations to determine the current state of the board. Usually such small timing differences are not distinguishable over a network. If instead, the timing difference between the cases when first k characters of the password are correct, and the first k + 2 characters are correct is distinguishable, then it would require a maximum of 11+434319+43+1 = 35; 188 active operations to do the same. Since there are 30; 000 allowable active operations, we conclude that if the timing differences are not distinguishable in the latter case, the application is not vulnerable under the operational budget. […]"

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable. Colorado performed analysis of Java's String.equals() method, note, any potential vulnerability discovered in this method would be counted but be noted as out-of-scope since the method is indiscriminately called to check all passwords. Additionally, while this does not impact the team's evaluation, their response only demonstrates an analysis of one avenue for attack through the password and does not indicate any analysis of the potential to map the target user's moves based on response time or analysis of potentially decrypting the state from the messages sent from the server to the target user. The provided assumption in the challenge question that the attacker knows the target user's username does not mean any possible attack requires the username, only that if a discovered avenue of attack requires the username then it can be assumed that the attacker has it.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"After discussions with Northeastern and Vanderbilt teams, we elaborate our justification; However, we do not change our final answers. We observe that there are timing side channels in the application that depends on the weight of pieces. For example, we observe that it takes more time for movements close to king. It takes about 15 seconds to move pawn in the front of king, 13 seconds to move pawn close to and 11 seconds for other pawns. As a result, an attacker who observes the sequence of timings related to these movements potentially could find out the current state of all pieces. However, the space of problem is quite large that makes it impossible in the given budget."

Live-Collaborative Evaluation:

Colorado correctly concluded that a side channel they found was insufficient for the program to be vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.6.2 Question 015 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Yes, there exists an algorithmic complexity vulnerability in time that causes a benign "new game" request to be delayed by more than 1000 seconds under the input budget. The game rules state that a pawn that reaches the last row on the opposite end of the board (from where it started) must be promoted to a rook, bishop, knight or a queen (but not a king.) The piece to which the pawn is promoted is given as part of the move request when the pawn is ready to move into a square in the last row as follows: move <origin square target square> <piece>. Although the rules do not allow the promotion of a pawn to a king when it reaches the last row, we show in

the next paragraph that there exists a vulnerability that allows for an attacker to do so. In the com.ainfosec.chessmaster.Server.serve method, the application checks whether <piece> == king, and if so gives an error. However, the Autocorrect feature is then called on the <piece> input, which checks whether a small modification of the word (such as adding a character, deleting a character or swapping two characters in the word) matches any of the words in the String array path. This String array contains the word "king" although it is not needed. Hence, if <piece> is a small modification of "king" such as "kin" or "kng", then it will pass the initial string equality check, but will then be auto-corrected to "king" anyway. In this way, an attacker can force the server to promote his pawn that reaches the other end of the board to a king piece and thus end up with two (or more) kings instead of one. It remains to show how the attacker possessing two king pieces can trigger the AC time vulnerability. The AI determines its next move by evaluating all possible moves from that game state onwards, not only for the next turn but also a few subsequent turns, and then determines its next move. The depth of turns from the given game state that the AI evaluates is determined by the integer "depth". This integer is a quadratic function of the integer x calculated in the com.ainfosec.chessmaster.State.eval() method. This method scans the entire board, assigns a weight to each piece depending on its type, and returns the difference between the sum of the weights for the white pieces and the sum of the weights for the black pieces. The weight assigned to a king (20,000) far outweighs the weight for any other piece. Hence if there are two white kings and only one black king, x and hence depth will be a very large integer. This forces the com.ainfosec.chessmaster.GameClient.negamax method to recurse a large number of times, because the number of recursions made by this method is controlled by depth. […] We were able to find this vulnerability by using CodeHawk to find and display all of the recursive methods in ChessMaster (Figure 11). Fortunately, there is only one and it contains our vulnerability."

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Colorado identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.7 Class Scheduler

A.6.1.2.7.1 Question 017 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"This program constructs a class schedule by implementing a genetic algorithm and permits the administrators to supply an upper bound on the number of generations between 100 and 1500. In order to delay the divergence of the implemented genetic algorithm, an administrator can provide a hard constraint for the algorithm to satisfy, for instance by providing school data for which no valid schedule exists. One way to accomplish this is by uploading an xml file with 12 teachers, 122 students and 29 courses with the condition that one teacher teaches all of the courses while

others teach no course and then requiring all of the students to have same 4 required courses. However, under such an input file it is not possible to make each generation run longer than 0:2 seconds. However, the application has a bug that allows the administrators to provide an input where some of the courses have a negative number of sections. This allows an attacker to provide an arbitrary large number of sections for the required courses, while bypassing the check that total number of courses allowable should be less than 29. This bug results in increasing gene size resulting in algorithmic complexity vulnerability. The vulnerability exists in method validateData of the file VaadinUI.java inside the directory com.bbn.classScheduler.scheduler. In this method, the check (totalCourseSections > maxTotalCoursesAllowed) passes since the number of sections is permitted to have a negative value."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The configuration of the server application used to handle incoming requests is believed to prevent this reported vulnerability; however, we do note that this reported vulnerability results in near 100% CPU utilization. During collaboration, R&D Teams identified an exploit to crash the server. An exploit is not required, and Colorado's take-home response exercises this unintended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"After sharing our exploit with the other teams Utah pointed out that the wording of the challenge problem requires there be a 50mins delay between the time of the request and the response, that is, any response, which was confirmed by Kwame. Since Vaadin is a non-blocking server it comes back with a very quick response, a message, even if, at the same time the server is computing the schedule, which we then also observed. We discovered another vulnerability that addresses this shortcoming in our exploit: the class com.bbn.classScheduler.scheduler.VaadinUI has a boolean fieldgenerationInProcess, which is supposed to ensure that a request for schedule generation is denied when a schedule generation is already processing. However, this field is local to the VaadinUI, and every client (new browser, or new tab in the same browser) creates a new instance of VaadinUI in the server, with the fieldgenerationInProcess initialized to false. There is no checking if other processes exists, so it is possible to start multiple threads in the server that generate the same schedule simultaneously. When we open five tabs in the browser and start the schedule generation from each of them, a subsequent request to download the schedule hangs, because the CPU is 100%occupied and (apparently) does not yield to a request from the server, rendering the server blocking during the remaining time of the schedule generation. Note: we only tried this on a Mac; we did not have time to try this on the reference machine. In a subsequent session Iowa State ran our exploit in the same manner, on a Mac, obtaining the same behavior. In addition, he tried to connect to the server with a curl command, which also blocked, confirming that itis possible to make the server unresponsive. The consensus was that this is probably caused by CPU exhaustion on the JVM, rather than exhaustion of threads, because the number of threads in use is rather low, while the CPU utilization goes to 100%."

Live-Collaborative Evaluation:

The challenge does not contain an intended vulnerability. The unintended vulnerability reported was confirmed. It is unclear what the initial part of the response is attempting to say, however I believe the clarification that was provided was that the challenge question provides the benign

user request to be delayed. When the benign user sends that request, the server should provide a response within the resource usage limit. Any identified vulnerability that causes the server's response to the benign user to be delayed at or beyond the resource usage limit is counted as a vulnerability. Note that the response can simply be that the server is busy.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.8 Effects Hero

A.6.1.2.8.1 Question 025 (AC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"Yes, there exists an algorithmic complexity vulnerability in time that would cause the real runtime of the benign request to exceed the resource usage limit given the input budget. The exploit is shown in the Figure 13. The vulnerable code is in the method process in file com.bbn.hero.BeatBlock.java that, for an appropriate combination of sample duration in seconds and BPM value of the BeatBlock effect, results in a large number of invocations of the fade subroutine of BaseBlock.java that, in turn, invokes expensive matrix multiplication (cubic) and cofactor (quartic) subroutines. The vulnerability was found because our tool CodeHawk flagged the matrix multiplication and cofactor subroutines as expensive procedures (Figure 14), and traced their invocation to the BeatBlock effect (Figure 15). After this discovery, we focused on the BeatBlock effect and experimented with several combinations of \sample durations" and \BPM" settings to discover the timing vulnerability. This vulnerability was then exploited by using \30" second sample duration (highest permissible) and BPM setting of \444444" for BeatBlock effect whose inputs were connected to default \FunctionalGeneratorBlock". In our exploit we use two blocks of the FunctionalGeneratorBlock effect, and 48 BeatBlock effects such that the application took more than 600 seconds for total input request size that was well below 2 KB."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The unintended vulnerability reported by Colorado was identified by several other teams, and we note that it results in high CPU consumption; however, as observed by Two Six Labs, this identified vulnerability is not believed to sufficiently impact the benign request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"No, the algorithmic complexity vulnerability in time is not exploitable in given budget. After the discussion with Vanderbilt and Iowa State, we decide to change our answers from Yes to No. The reason is the multi-thread behavior of effects hero that does not block the benign users to create new sessions. Since there are possibility to create 200 threads and each of our attack scenario requires about 400 KB, the exploit will exceed the given budget."

Live-Collaborative Evaluation:

Through collaboration, Colorado correctly concluded that the program was not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.9 Railyard

A.6.1.2.9.1 Question 013 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"No, there is no algorithmic complexity vulnerability in space. The arguments are that

1. Although an attacker is able to create a cycle in the linked list of cars and the traversal could in theory lead to infinite write to a StringBuilder object (See answer in Question 19), the piece of code infinitely writing to StringBuilder object is actually unreachable because the infinite write to disk need to terminate (which cannot happen) before reaching the piece of code infinitely writing to StringBuilder object and the path constraints are unsatisfiable to skip writing to disk and jump directly to writing to StringBuilder object. CodeHawk shows (Figure 16) that the file write dominates the StringBuilder write on the control flow graph when the variable fileStream is not null which is indeed true because the attacker has no control over file read/write on the server.
2. All collection-typed data structures in the application have linear memory usage in terms of input sizes. There are several major collection-typed data structures in the application […]"

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Yes, there is an algorithmic complexity vulnerability in space. During take-home Engagement, we thought of an exploit that works in theory that takes advantage of race condition of the application, but we did not manage to get an exploit that works in reality. Out of conservative thoughts, we did not mention it in the previous answer. However, during Collaborative Engagement with collaboration with people from Vanderbilt, Iowa and GrammaTech, as a team we managed to come up with an exploit that works in reality, which is an implementation of our (i.e. Colorado team's) theoretic exploit during take-home Engagement. The vulnerability stems from the static field of classCoal Car (which is the same as Question 19): We can create a loop of linked cars by adding two coal cars. Then upon user request sendout, the application will1(1) invoke method printToStream to traverse the linked list and write every list element to disk and then (2) invoke again method printToStream to traverse the linked list and write every list element into a string that resides inmemory. […] The key of this exploit is the race condition. The exploit works only under the following situation: The loop in the linked list is re-established exactly when Thread A is invoking method printToStream for the second time."

Live-Collaborative Evaluation:

Collaborating with Vanderbilt, Iowa, and GrammaTech Colorado appears to have flagged a potential unintended vulnerability. We believe that this vulnerability is indeed present in the application.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.9.2 Question 018 (AC Space, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“Yes, there is an algorithmic complexity vulnerability in space. The exploit is under the directory E6-solutions/exploit 018. The vulnerable code is the field declaration class railyard.cars.Coal Car. The next field of class Coal Car is declared as static, while all other car classes (e.g. Mail Car) do not declare their next field as static. This is vulnerable because an attacker could take advantage of static field to create a cycle in a linked list | Upon user request send out, the application will traverse the linked list and write every list element into disk. Therefore, a cycle in the linked list leads to infinite write to disk. […] CodeHawk indicates (Figure 17) that the next field in class Coal Car is static and the writers to this field is in the class initializer, indicating that the next field is never written by any member methods. We found this suspicious because it is unusual that a static class field is only written at the initialization stage and never updated anywhere else. Later we realized that the next field is updated through Java reflection.”

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.10 SimpleVote 3

A.6.1.2.10.1 Question 001 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

“Yes, there exists a side-channel in space and time from which an attacker can obtain any single ballot. The vulnerability is possible because any user that can login the system can access any election by simply visiting the URL https://serverNUC:8443/ballots/XXX with XXX being the digits of the selected election. After logging into the system, the source code of the overview page shows the digits for each election. After accessing the election page via the above-mentioned URL, the user is asked to enter a registration key, which is unique for each ballot. Entering the registration key will allow the attacker to access the ballot (without requiring any login details of the target voter). The attack consists of two parts: 1) obtaining the registration key of the target user using a side-channel in time; and 2) determining whether the target voter accessed a ballot using a side-channel in space. Regarding the side-channel in time: The registration key consists of two parts, each of 6 characters. Let's call them the matching part and the index part. One can permute the characters in such a way that the first 6 characters belong to the matching part and the second 6 characters belong to the index part. This is also what happens in the application when the registration key is validated. The runtime of the validation depends strongly on the index part of the registration key, see Figure 18. The larger the index part, the higher the validation time. The difference between two consecutive keys, the runtime difference is roughly 0.03 seconds, which makes it easy to pinpoint the key. Regarding the side-channel in space: The target voter may enter the wrong registration key during the session. The attacker can

easily see whether the registration key was correctly entered due to a side-channel in space: The page shown after incorrectly entering a registration key is much smaller than the page shown after accessing a ballot (using the correct registration key).

A strategy for obtaining the target ballot is as follows: First, use the passive operation to obtain the timings and sizes of the packages of the target voter. The packages with the highest timings are related to validating the registration key. A large package means that the validation is successful. Since voters are typically only involved in one or two elections, there will be only one or two of such packages. Based on the timing of the package, one can obtain the index part of the key. Let t denote the time of the packages. The index part is 0 (mod 7) and roughly $0.03 * t$. The matching part can be computed by the attacker using the logisticMapping procedure as

each observed timing uniquely identifies the registration key (in contrast to the simplevote applications from Engagement 5). Afterwards the attacker uses the computed registration key (possibly two) to test whether it is valid for election with id XXX by posting the registration key on the page https://serverNUC:8443/ballots/XXX. This will require at most 12 active operations for a single key, because there are 12 elections. In case of two registration keys, the strategy is applied for the first key. After successfully determining the election, one oracle operation is applied to ask whether the right ballot has been found. If that is not the case, then the same procedure is performed for the second key for the remaining 11 elections. […]"

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. We do not believe the challenge is vulnerable as while part of the key is leaked by the reported vulnerability, we do not believe that there are sufficient operations to map the leaked partial key to the full registration key. In this version of SimpleVote, the index i of the registration key does not uniquely determine its other part. Rather, the string corresponding to any $j <= i$ can be combined with index i. So, in the worst case, there are more than 100 keys that correspond to a given time. With only 25 operations, no exploit is possible.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We changed our answer during the collaborative engagement to no. The attack described in our original response required several elements. The first one is obtaining the registration key. All teams agreed that there is a side channel in time that allows obtaining the key (this was not possible in other simplevote variants). The second part of the attack is being able to access a ballot with only the registration key. Although this was possible in other simplevote variants, this is not possible in simplevote3. The Vanderbilt team showed us that theEasyDecisionServiceImplementation.checkRegistrationCodeprocedure checks whether the key belongs to the attacker. This is not the case. The other teams used a similar argumentation why that attack would not work."

Live-Collaborative Evaluation:

Through collaboration, Colorado correctly concluded that their proposed exploit was invalid.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.10.2 Question 034 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“No, there is no exploitable side channel in space from which an attacker can discover the percentage of finalized votes received by the target candidate. There exists a very weak side channel in space in easydecision.manager.EditBallotManager.legitimizeChoices: the attacker is able to update the INCLUSION THRESHOLD via the method updatel by filling in the name of the target candidate in the free-form field proceeding with exactly 2 space and followed by n spaces. The INCLUSION THRESHOLD determines whether the package of the target candidate send during synchronization is of stand size or smaller. In the other simplevote variants this side channel is strong and allows determining whether the target candidate has at least n percentage point of the finalized votes. In this variant however, the side channel is weak due to the missing factor of 100. As a consequence, the side channel only reveals whether the target candidate as 100% of the vote or not. This however can also easily be determined by a single oracle query.”

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.10.3 Question 037 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

“Yes, there is a side channel vulnerability in space from which an attacker can obtain the password of a target voter. The password checking is performed in the procedure webcontroller.manager.LoginManager.handleParcel which calls TokenedPasswordChecker followed by processPassword. The latter compresses the actual password and the provided password in such a way that there is a side channel in space: more matching characters between the actual password and the provided password results in a smaller package. The side channel can be observed using wireshark: the size of the first returned package after submitting the password reveals how many characters of the password are correct. This allows the following attack: try all passwords of a single character and select the character for which the returned package after submitting the password is the smallest one. This is the first character of the password. Repeat this procedure to obtain the second, third, etc. characters of the password until the login is successful.”

Take-Home Evaluation:

We don’t believe this reported side channel is vulnerable as the size of the packets are not consistent between different guesses of the same password.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“After discussions with the Vanderbilt team, we have concluded that the exploit described below does not achieve the desired result within the budget, we justify our change below. The side

channel vulnerability mentioned in the original response is too weak to exploit within the given budget of 1500 operations. Passwords can be 15 characters long and the character range is in between "!" and " ", meaning a range of 90 characters. In other words, the side channel has to be very strong to allow an attack within the budget, so there is very little room for noise. However, the side channel is not very strong, especially for the later characters in the password (say, after position 10). The weakness in the side channel is due to the randomness added in the TokenedPasswordChecker.processPassword procedure."

Live-Collaborative Evaluation:

Through collaboration, Colorado's potential exploit was disproven and the correctly concluded that the program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.10.4 Question 038 (AC Time, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"No, there is no such algorithmic complexity vulnerability that could be invoked by requests with the total of 4kB PDU. Although there exists a vulnerability that could cause the runtime of a benign request to exceed 300 seconds, but invoking it requires more input budget than 4 kB. The part of the procedure com.cyberpointllc.stac.mathematic.LogisticMapping.logisticMapping shown below shows a loop in which the procedure followingCore is being called. The followingCore procedure calls the procedure combine which runs a nested loop that results in doubling the precision of returned value and thereby roughly doubling the runtime in each call. Hence, the loop has a worst-case exponential complexity and the user can control the loop length (parameter n).

```
BigDecimal x = new BigDecimal(x_0);
...
for (int k = a + 1; k <= n; ++k) {
x = LogisticMapping.followingCore(x);
}
```

Although parameter n is user-controlled, it is restricted: The call to logisticMapping – which is called from the procedure checkCore - is sanitized by the method isInBounds, which limits n to be at most MAX N = 995 initially. Moreover, the challenge application consists of a table that sets the k (the start of the loop) to the index of the last valid registartion key, which is at most $n - 6$. However, it is possible to increase MAX N by one by calling isInBounds with MAX N although it returns false and therefore blocks calling checkCore. Increasing MAX N also breaks the invariant that the maximal difference between n and k is at most 6. Also, there is another procedure checkSurface which checks if n is divisible by 7. Since checkCore is called only if checkSurface returns true, therefore the procedure checkCore is executed only by values of n which are divisible by 7. We can reach the runtime of 300 seconds by increasing the MAX N to 1001, which the first number greater than 995 that is divisible by 7. In order to that, we need 7 requests. Then, we can send a request with n = 1001, which bypasses the sanitations and invokes the procedure checkCore. It also breaks the invariant that the maximal difference between n and k is at most 6 in the procedure logisticMapping. When logisticMapping is called with n = 1001, it finds the last valid registration key at index 987. Therefore, it runs the above-mentioned loop

14 times, which causes the runtime to exceed the 300 seconds limit. This exploit requires at least 8 requests from the attacker, because the MAX N should be increased to 1001 in order to bypass the sanitization code. Since each request has the PDU size of about 1 kB, exploiting the existing vulnerability is not possible with the given input budget."

Take-Home Evaluation:

The challenge program did not contain an intended vulnerability; however, Utah and Northeastern identified the same potential vulnerability as Colorado during the take-home and Northeastern helped other teams during the collaborative engagement to craft an exploit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We changed our answer during the collaborative engagement to yes. The algorithmic complexity vulnerability that we found during the take home engagement is used. However, our initial assessment that the budget was too small is incorrect. The Northeastern team showed us how to significantly decrease the package size. The exploit script provided by Northeastern […] reduces the package size from roughly 8kb (more than the budget) to slightly more than 3kb (less than the budget). It blocks benign requests for about 450 seconds on the serverNuc (i.e., more than the required 300 seconds)."

Live-Collaborative Evaluation:

Collaborating with Northeastern, Colorado responded with an exploit that fits within the input budget. This unintended vulnerability was confirmed within the budget.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.11 SimpleVote 4

A.6.1.2.11.1 Question 023 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"No, there exists no side-channel in space and time from which an attacker can obtain any single ballot. This attack requires obtaining the registration key of the target voter. However, the side channel in time regarding to check the registration key is too weak to uniquely determine the registration key. There are 142 different keys in the database and Figure 21 shows the weak relation between a key and its validation time. The lack of a strong side channel is due to the absence of a triple nested loop (which is present in the other simplevote variants). In particular the confirmValue procedure is not as costly as the corresponding procedure in the variants as it calls the expensive method (in this case chaoticOperator) only once (instead of n times). […]"

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.11.2 Question 030 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Yes, there is an algorithmic complexity vulnerability that could cause the runtime to exceed 300 seconds. This vulnerability can be invoked by three requests from the attacker which requires less than 4 kB PDU overall. The procedure com.cyberpointllc.stac.math.ChaoticOperator.chaoticOperator is very similar to procedure com.cyberpointllc.stac.mathematic.LogisticMapping.logisticMapping in the application simplevote3. As shown below, chaoticOperator executes a loop in which the procedure nextValue is being called. The nextValue procedure calls the procedure join which runs a nested loop that results in doubling the precision of returned value and thereby roughly doubling the runtime in each call. Hence, the loop has a worst-case exponential complexity and the user can control the loop length (parameter n). […] Although parameter n is user-controlled, it is restricted: The call to chaoticOperator – which is called from the procedure confirmValue - is sanitized by the method isInBounds, which limits n to be at most MAX N = 995 initially. Moreover, the challenge application consists of a table that sets the k (the start of the loop) to the index of the last valid registration key, which is at most $n - 6$. However, it is possible to increase MAX N by one by calling isInBounds with MAX N although it returns false and therefore blocks calling confirmValue. Increasing MAX N also breaks the invariant that the maximal difference between n and k is at most 6. Although this part of the simplevote4 application is very similar to its corresponding part in the simplevote3 applications there is a subtle difference that enables the attacker to invoke the existing vulnerability with only three requests. Similar to the method checkSurface in simplevote3, here procedure confirmExternal checks if n is divisible by 7. However, the confirmExternal is executed after the procedure confirmValue, so it does not prevent the execution of confirmValue with values of n which are not divisble by 7. This enables the attacker to exploit the vulnerability in the application by sending only three requests as described in the next paragraph. We can reach the runtime of 300 seconds by sending three requests with n = 995. The first request increases the MAX N to 996. This request does not result in executing the confirmValue method, because isInBounds returns false. The second request with n = 995 bypasses the sanitization in isInBounds due the increased value of MAX N and executes confirmValue method. This execution of confirmValue adds an entry to the last position in table, which causes invoking the method com.cyberpointllc.stac.math.Stash.composeOpening. The procedure composeOpening sets the last 10 entries in the table to null. Therefore, the third request with n = 995 sets the k to 880 in the method chaoticOperator, because now the last valid registration key is at index 980. This breaks the invariant that the maximal difference between n and k is at most 6 and results in running the above-mentioned loop 15 times. Therefore, the runtime exceeds the 300 seconds limit as the loop is exponential. Since the attacker only needs to send three requests, the total PDU size is less than 4 kB.

The vulnerability was found because our tool CodeHawk flagged the nextValue as a high-cost procedure. CodeHawk additionally traced execution from the source of handling a HTTP request to the method chaoticOperator which uses the vulnerable nextValue procedure. This confirmed that user input over the network was able to influence the execution of the vulnerable method chaoticOperator. […]"

Take-Home Evaluation:

Colorado identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.12 STAC Coin

A.6.1.2.12.1 Question 008 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)*

Take-Home Response:

“No, there is no algorithmic complexity vulnerability in space. The arguments are that the only method that writes to disk is Persistence.writeWalletToDisk, method Persistence.writeWalletToDisk serializes an instance of class Wallet into disk and 3 in the

Control flow graph, all five call sites to method Persistence.writeWalletToDisk are dominated by a size increase of class Wallet that is linear to input sizes. Next, we justify why the size increase of class Wallet is linear to input sizes for each interesting user request:

- When processing user request newBlock, the application invokes method BlockChain.extend and the size increase of the instance of class Wallet is composed of two parts: Size increase of the instance of class BlockChain together with size increase of the instance of class Ledger, because class BlockChain points to class Ledger and class Wallet points to class BlockChain. […]
- Equivalence between new transaction's size and class Wallet's size increment Class Miner indefinitely monitors every update in queue txQueue. Whenever a new transaction is appended there, class Miner will mine for a random number such that the hash value of a new block has at least 4 leading 0's, where the new block contains the random number, hash value of the previous block and the new transaction. There are in total two miners in our setting, competing with each other on who first finds out a correct random number. As long as one of the miners finds a correct random number, it will invoke method BlockChain.extend to append the newly generated block to list blocks in class BlockChain, leading to a size increase of the instance of class Wallet where the amount of increase is linear to the size of the new transaction (this is justified in the case of user request newBlock, which also invokes method BlockChain.extend).
- New transaction's size Recall that a transaction is a combination of a list of Input and a list of Output and an Output is a combination of a recipient address and an integer. An Input consists of a transaction ID of an earlier transaction, an index specifying an Output in that earlier transaction and a private key of the transaction initiator. To sum up, the sizes of either class Input or class Output are constants. Hence, it remains to be determined the lengths of lists to compute the size of the new transaction. […] To sum up, upon every user request spend, the length increase of collection-typed data structures

reachable from the instance of class Wallet is at most 2, leading to a size increase linear to input size."

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Yes, there is an algorithmic complexity vulnerability in space. The key is that the exploit fits into the input budget only under the assumption that the size of a response is not counted in the input budget — only the size of a request counts. The exploit works as follows:

1. The attacker locally creates a very long block chain by generating lots of transactions (e.g. transfer money from one address to the same address)

2. Trigger the other client to request updating its own block chain under the assumption that it has an old block chain. Then the attacker will send the very long block chain to the victim client, who is going to append the very long block chain and store it locally, thus reaching the resource usage limit."

Live-Collaborative Evaluation:

As correctly noted by Utah in their response, the input budget applies to the total bytes sent from the attacker to the device with the application under attack. We note that Colorado interpreted that bytes sent in response did not count towards the budget. Therefore, this response was treated as a "No" response as the assumption under which it was made was incorrect.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.13 Swappigans

A.6.1.2.13.1 Question 002 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The attack proceeds as follows. The attacker registers and logs into the server with any new user. The attacker then adds 29 new items. These items have prices that sum up to near the maximum price $1000.00, specifically the price of the most expensive item. The price of each item is half of the price of the preceding item, with a minimum price of $0.01, and the items are added in decreasing price order. After the items are added the attacker sends a purchase item request for the most expensive item. This causes the swappigans application to hang, causing a denial of service until the request processes. […]"

Take-Home Evaluation:

The challenge contains an intended vulnerability that requires the attacker to exceed the maximum price. Colorado identified an unintended vulnerability in the number of items, price, and price difference in individual items.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.13.2 Question 020 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“No, the swappigans application does not have a side channel vulnerability in space and time that allows them to make a purchase using the target user's items. The swappigans application was inspected to investigate this possible side channel. There were three possibilities to allow this attack: the attacker knows the target user's username and password, the attacker knows the target user's username and session key, or the attacker is able to get the target user's items into their list of items for consideration in purchase. The application does not have a side channel that would allow the attacker to get the target user's password given their username. The password is transmitted directly without padding to the server as part of the login GET request. The request is: https://serverNuc:8080/login? userName=<username>&password=<password>. With this request there is a side channel in space that allows the attacker to know exact length of the password. The length of the password is the length of the packet corresponding the login request minus 203 bytes. No side channel in time was observed. Inspection of the code shows that the application hashes the password plus a salt using the MD5 hash. The hashed password is compared against the stored hash. Since MD5 is a one-way function and the block size for MD5 is 64 bytes it is not possible to use a time side channel to infer the password. However, the users.csv file contains the list of users and their password hashes and associated salts. It would be possible to brute force the password online, since the salt and the hash are known. But with password lengths up to 32 characters this is not a reasonable approach, even with the known password lengths from the space side channel. A bytestring of length 32 up to $(2^8)^{32} \approx 1E77$ possible options.

The application does not have a side channel that would allow the attacker to infer session token. On login, the server gives a session token to the user, and this session token is used to authenticate the user for all following interactions. The session token is generated by using a server private key to encrypt the username when the user successfully logs in. If the attacker were able to produce the session token, then they could impersonate the target user without their password. The key is a private static field of the SessionTokenManager class and is generated from a random string of 36 possible characters with length 16 ($8E24$ possible options). The random generation of the string is seeded using rnd.setSeed(System.currentTimeMillis() / 1000L). If the time when the seed was generated was known, then the private key could be inferred. However, since this field is initialized at the startup of the server, there is no way to deduce this time from a side channel. Furthermore, the session tokens are generated using an AES encryption operation. The output of the AES cipher is one way and thus the session token is not able to be spoofed using a side channel. Finally, the application does not have a side channel that would allow the attacker to use another user's items. The only items that are used in the purchase method are those registered the verified user in their itemsForTrade field. Only a user verified by a correct session token is able to add items to this field.”

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“All other teams presented a vulnerability that involves obtaining the master key from the time that the server was booted, which is possible, because the search space is made small relative to the number of operations available (it allows for the server having been booted decades ago). Every possible instance can then be tried as the seed for the master key, evaluating if the encryption gives the expected result. Everyone seemed to agree that this is not really a side channel in time (although time is involved), so we decided to keep our original answer that there does not exists a side channel vulnerability in time.”

Live-Collaborative Evaluation:

Colorado correctly concluded that the challenge program did not contain an in-scope vulnerability. The vulnerability discovered by other teams was counted as a valid out-of-scope vulnerability that leaks the secret without a side channel.

Post-Engagement Analysis Ruling: Correct

A.6.1.2.13.3 Question 031 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“No, the swappigans application does not have a side channel vulnerability in time that would allow the attacker to determine which item was purchased by a target user. As described in the response to Question 02, there is an algorithmic complexity vulnerability in time found in the method SwappigansItemMatcher.calc(). The run time of this method is determined by the items that a user has and is positively correlated with the price of the item being purchased. An attacker is able to impersonate the target user in terms of the inputs to the calc() function. The attacker can register a new user with no items and add a list of items that matches the items of the target user, with the same item ordering. By doing this, the time behavior of the server will match that of the target user when the attacker purchases an item owned by neither. This is because the calc() function has the same inputs. By doing this the attacker can determine the server timings for each possible item a target user is able to purchase. After generating a table of timings, the attacker can observe the target user purchasing an item and compare that time to the table. However, for this to be feasible there must be observable timing differences between classes of items. The timings table can be generated online if there is access to the server. Then one operation can be used to measure the timing of the target users purchase request. This gives the attacker information about the price of the target item, and leaves 31 operations that can be used for oracle queries. Note while there are many items that have almost the same price, there is no set of items with almost equal price that have more than 31 members. The timing on the calc() function only increases noticeably when enough different price items are used to purchase an item and if the price of the item is not low. This is not the case for all the possible target users. The user CRACCHIOLO has 28 items and the timings of their emulated purchases show a clear separation in figure 26. Based on experimental data, the attack would work with 97% certainty for this user. When the items are clustered into families of almost the same price then there is still clear separation. However, there are only two users, CRACCHIOLO and SHUMARD, who have a proper set of items to create such timing separation to achieve an attack with an 85% probability of success. Every other user does not have discernible timing differences.

Experimental results for the next best user determined a 47% chance of success using 31 oracle queries. Figure 27 shows the timings without sufficient separation. To illustrate this, we apply our tool Discriminer with constraint-based clustering. In this case, we specify the constraint that the distance between the centroid of two cluster should be at least 2 ms. In this case, Discriminer could find 3 clusters as shown in Figure 28. As each cluster corresponds to one classes of observations (partitioning the set of secret values to the classes of observations where elements of the same cluster are indistinguishable), there are 3 classes of observations for this application. The secret price can be observable for any value if we could find following number of classes of observations: number of secret values * 0.85. In this case, we consider 728 different secret item price and observe only 3 classes of observations."

Take-Home Evaluation:

Colorado correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.2 GrammaTech

A.6.2.1 GrammaTech Overview

GrammaTech was the only team to answer all 43 challenge questions, and achieved the third highest accuracy of 74%. Iowa, GrammaTech, Colorado, and Vanderbilt all achieved between 73% and 76% accuracy in the take-home, but GrammaTech answered 3 more questions than the next team, Vanderbilt. The TPR and TNR metrics serve to better distinguish the performances of these four teams. GrammaTech achieved an 89% TNR and a 53% TPR, matching Iowa to within 3 percentage points. In the leading cluster of the 4 R&D Teams, GrammaTech achieved the third highest TPR and TNR, being marginally outperformed by Iowa in the segmentation of answered questions. Against the segmentation of all questions where unanswered questions are treated as incorrect, GrammaTech achieved the highest TNR of all performers and the 2nd highest TPR of the R&D Teams.

GrammaTech had the most difficulty with AC Space questions during the take-home (50% accuracy) with the team achieving 100% accuracies in SC Time and SC Time/Space questions. In our analysis of the reasons for incorrect responses, in SC questions, GrammaTech's incorrect responses were due to failing to identify the side channel or incorrectly assessing the strength of identified side channels. In the case of AC questions, the team's incorrect responses were due predominantly to not understanding the input to the complexity control flow required to exploit a vulnerability, motivating the team to incorrectly declare the challenge non-vulnerable. With the number of questions answered and performance across all categories relative to other R&D Teams, GrammaTech was one of the top take-home performers in Engagement 6.

During the live-collaborative engagement, all R&D Teams achieved between a 97% and 100% accuracy, with all teams achieving a 100% TNR in the segmentation of answered questions.

GrammaTech tied with Northeastern and Vanderbilt for the second highest accuracy of 98% with the team only missing the complex distributed SC Space vulnerability in the Case DB challenge program. Collectively, the R&D Teams identified 3 unintended vulnerabilities during the live collaborative engagement - Railyard Question 013, Class Scheduler Question 017, and SimpleVote Question 038. Railyard Question 013 and Class Scheduler Question 017 came out of a collaborative effort from all teams. The SimpleVote Question 038 unintended vulnerability was confirmed with a clever exploit by Northeastern.

A.6.2.1.1 Take-Home Engagement Scores

Table A-121: Engagement 6 GrammaTech Take Home Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	5	62
SC Time	12	12	100
SC Space/ Time	5	5	100
AC in Space	8	4	50
AC in Time	10	6	60
Total	**43**	**32**	**74**

Table A-122: Engagement 6 GrammaTech Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	16	9	53
Not Vulnerable	26	23	88
Yes Answer	13	9	69
No Answer	30	23	77

A.6.2.1.2 Live-Collaborative Engagement Scores

Table A-123: Engagement 6 GrammaTech Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	7	88
SC Time	12	12	100
SC Space/ Time	5	5	100
AC in Space	8	8	100
AC in Time	10	10	100
Total	**43**	**42**	**98**

Table A-124: Engagement 6 GrammaTech Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	17	16	94
Not Vulnerable	26	26	100
Yes Answer	17	17	100
No Answer	26	25	96

A.6.2.2 GrammaTech Specific Responses

A.6.2.2.1 BattleBoats 3

A.6.2.2.1.1 Question 022 (AC Space, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"When a player makes a shot, it sends a SHOT_MADE message of the form: shoot <h> <v> <elevation angle> <board angle>. The opponent receives this message and computes the squares that are hit by the shot. Part of this computation happens in the method com.cyberpointllc.stac.battleboat.HitPinpointer.determineHitTime(), which contains the algorithmic complexity vulnerability. As in the battleboats examples from Engagement 5, this method iteratively computes the time the shot hits the ocean. Because this code manipulates BigDecimal objects, and memoizes some of these BigDecimals in a HashMap, the memory consumption of the method can be significant for certain shot parameters."

Take-Home Evaluation:

GrammaTech identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"During the collaborative discussion, Colorado demonstrated an exploit similar to ours which gave >4GB memory usage when run. Consequently, we remain convinced that a vulnerability exists and is exploitable. The remainder of the discussion attempted to pinpoint the exact cause of the vulnerability. The belief was that the size of the HashMap itself could not account for the excessive memory usage; therefore, the usage may have been due to slow garbage collection and/or inconsistent usage of the MathContext objects in the numerical computation code.

The difference between our attack and Colorado's attack is that they are using tiny numbers for all the parameters of the shot, while we are using a small parameter for the initial velocity only. Later, a member of the Iowa State team made further experiments and confirmed empirically that the most "important" parameter was indeed the initial velocity, which confirms our own observations from the take-home.

He performed some taint analysis of this parameter and suggested the high memory usage might be in the method terminate, that calls sqrt with an argument that is built with initSpeedY.pow(2,mc),which is going to have a large size. Inside sqrt, there is a creation of a large bigdecimal new BigDecimal(Math.sqrt(x.doubleValue()), which is a possible reason for the

heap size explosion. However, because this call to terminate is only executed once, we are unsure if this is the right justification."

Live-Collaborative Evaluation:

GrammaTech was more confident in their answer following collaboration, expanding on the cause of the vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.1.2 Question 026 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret: The observables are the size of the network packets. The method of interest is com.cyberpointllc.stac.battleboat.BattleBoats.sendMessage(byte[] msg) because this is the method that generates network traffic. More specifically, the size of the message msg is observable.

Statements on any potential side channels within the application: There would be potential side channels if the msg argument of a call to sendMessage depended on the boat positions.

Statements on why the application does not contain a side channel (strength and reachability): We can analyze the way each message is constructed:

- GAME_OFFER depends on the game parameters, but not on the player's boats, as they are not even placed yet at this stage.
- GAME_ACCEPT, GAME_DECLINE and I_WON have a constant size
- BOATS_PLACED has a dependency on the cannon position, but not on the player's boats.
- SHOT_MADE depends on the parameters of the shot, but not on the player's boats or cannons
- ERROR can have different sizes but has no dependencies on the player's boats
- HIT_RESULT leaks some information about the player's boats. When receiving a shot, the list of squares touched by the shot is returned with its corresponding Pin: HIT, MISS, or the name of the boat. However, this leakage is normal in the game and does not provide extra information that could be used to discover the coordinates of all boats.

How our tools supported this conclusion: IST identified all the program locations that create network traffic. From there, manual inspection of all these locations using the CodeSonar UI led to the conclusion."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.1.3 Question 040 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Statements on if and how the observables relate to the secret: The observable is the time between the packets sent and received through the network. We understand the secret as being the field private final List<Boat> boats of the class com.cyberpointllc.stac.battleboat.OceanScreen, and private final Map<String, List<Square>> placedBoats in com.cyberpointllc.stac.battleboat.Tournament.

Statements on any potential side channels within the application: There could be an SCL in time in the application if the computation time of the shot results depended on the secret fields described above.

Statements on why the application does not contain a side channel (strength and reachability): The field placedBoats is only used during the initial phase of the game, when the user places the boats and cannon. During that phase, there is no network traffic until the user has finished placing everything and sends an "end_placing_boats" message. This means there can't be a side channel leak from this secret. The field boats in OceanScreen is only used during the game by the two following methods: […] Neither of these two methods appear to leak secret information about the position of the boats:

- The execution time of the method tagHitSquare has some extra calls to Map.put in case the shot hits a boat, but this does not leak secret information.
- The method updateHitsAsSunk also has a longer execution time if the shot hits the boats, but does not reveal more information than just the number of hits.

How our tools supported this conclusion: We manually identified the secrets in the application. Next, we used IST's detection of methods that create network traffic and CodeSonar's user interface to establish the absence of a relationship between secrets and observable.”

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program was not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.1.4 Question 043 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Statements on if and how the observables relate to the secret: After a user has placed their boats and the opponent's cannon, a BOATS_PLACED message is sent to the opponent. This message contains a field populated by the result of this.battleBoats.filter(this.battleBoats.obtainArrangementMessage()) […] The filter method actually takes a String as input and returns its length (as a String), and is applied twice, because it is also applied in obtainArrangementMessage: […] Moreover, pullArrangementMessage calls

the pullEncoding method : […] The value of encoding returned by this method leaks the position (x, y) of the cannon, because it's a String containing $23 * y + x$ times the space character. Also, its execution time is influenced by the values of x and y, i.e. the secrets.

Statements on why the application does not contain a side channel (strength and reachability): Because of the call to Math.random() in the inner loop of the pullEncoding method, we believe there is no SCL in time that would allow the attacker to guess the coordinates of its own cannon. There could be another side channel leak when receiving a shot, because the computation of the hit squares depends on the cannon's position. However, there is no significant time difference in the computation that would depend on the secret.

How our tools supported this conclusion: We used IST to identify the locations in the program that can generate network traffic. Then, manual inspection of the code led us to the answer."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Even though we did not mention in our take-home justification, it might be worth mentioning that there could also be a potential SCL in time when the attacker makes a shot and waits for the response. The Northeastern team pointed out that the computation of pullHits consumes some time and depends on the cannon position […] Also, this method is not always executed: the pullOceanSquares method can return an empty hashmap without executing pullHits in certain circumstances: when the shoot hits a position that is not on the ocean screen (which could potentially be useful to create an attack similar to the past engagement with a binary search strategy to guess the coordinates of the cannon) AND, the shoot has to be too far

[…] However, because BOTH conditions have to be satisfied, that means the pullHits method will still execute if the shot is not too far, even though it is out of the ocean screen. This makes the binary search attack more difficult. Now, if we look at the pullHits method specifically, it contains a number of if-then-else depending on the values […] and this is going to slightly influence the execution time because the number of calls to tagHitSquare will change. However, based on empirical results from experiments run by the Iowa State team, we believe this is far insufficient to be an exploitable SCL in time

Live-Collaborative Evaluation:

GrammaTech's live collaborative response flagged a potential side channel noted by several teams in their take-home response. The team correctly concluded that this side channel was too weak to result in a vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.2 BraidIt 3

A.6.2.2.2.1 Question 004 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Statements on any potentially vulnerable code structure in the application:

We have identified a vulnerability in the method com.cyberpointllc.stac.plait.Plait.freeNormalize(). This method makes a call to Plait.logger.info that writes to the disk, and can be executed an infinite number of times by the method com.cyberpointllc.stac.plait.Plait.normalizeCompletely(): […] The application can enter an infinite loop when the method com.cyberpointllc.stac.plait.Plait.isReduced() always returns false, which can happen for some well-chosen braids. Similar to braidit_2 from the past engagement, calling this method with the braid "Abcdefghijklmnopqrstuvwxyzzzzzzzzzzzzzzzzzzzzzzzzzzz" never terminates. This is because the braid starts with an A, and all alphabetical characters appear once except for z. This makes the return value of the method computeLoss be 26, and then the value of ground in isReduced() smaller than lowest: […] Because the braid does not contain a lowercase a, this method will then always return false. […]

Why the vulnerability is not exploitable:

The length of the above braid causing the infinite loop is 53, which is above the greatest allowed braid's length according to the variable com.cyberpointllc.stac.plait.Plait.HIGHEST_LENGTH. Even though this check is not performed when creating a Plait object, it is performed when the other player receives the MODIFIED_BRAID message in the method com.cyberpointllc.stac.plaitme.PlaitMeDispatcher.handleReceivedMessageHome: […] This means that, even though there is an apparent vulnerability, it does not seem to be triggerable by an attacker playing the game. An alternative idea would have been to "split" this braid in two pieces, such that it is constructed by a call to com.cyberpointllc.stac.plait.Plait.concatenate() when checking the equality of two braids. However, the loss of the concatenated braid is not the loss of the concatenated intersections, but only the loss of the input argument of the method: […] Again, this idea does not allow the attacker to trigger the vulnerability, as the return value of computeLoss can't be great enough.

How our tools supported discovery of the vulnerability:

The method com.cyberpointllc.stac.plait.Plait.freeNormalize() was idendified by IST as containing outputs (with the call to Logger.info) and formed a subproblem with the methods using it, e.g. com.cyberpointllc.stac.plait.Plait.normalizeCompletely(). We used that information, as well as manual analysis of the code and results from the past engagement, to reach our conclusion."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability; however, Iowa and Two Six Labs identified an unintended vulnerability in attacker-constructed braids that exceed the resource usage limit to process. GrammaTech noted that they identified a potential vulnerability that was not triggerable by an attacker playing the game. Note: that the attacker is not restricted to playing the game. The attacker can send any network request that the benign user will accept and process, therefore while the benign user's application generates and sends "random" braids, the attacker can craft and send any message to the benign user's application. In this instance GrammaTech correctly noted the enforcement on braid length for incoming messages, therefore the statement above simply serves as clarification for future engagements.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“In our collaborative discussion, Iowa State demonstrated an exploit which caused the program to exceed the space budget. The input is reproduced as the first argument in the method call below, which is part of Iowa State’s attack script: […] Although we have no deep understanding on how this input was chosen or why it causes excessive space consumption, we accept that it is a valid exploit; consequently, we are changing our answer to "yes".”

Live-Collaborative Evaluation:

GrammaTech admitted they did not fully understand the reasoning behind the vulnerability presented by their collaborator, but were able to see it work and were convinced enough to change their answer.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.2.2 Question 027 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Statements on if and how the observables relate to the secret:

We understand the secret to be a number between 1 and 5, representing the number of the braid which was modified. This is stored as field plaitIndex of com.cyberpointllc.stac.plaitme.phase.PlaitSelectedPhase. The value of this field is set when the user runs the select_braid command. The observable is the size of network messages. The observable relates to the secret because the value of plaitIndex is used as a parameter to specify the width of the string representation of the modified braid.

Statements on any potential side channels within the application:

The relevant code snippet is as follows, found in com.cyberpointllc.stac.plaitme.phase.PlaitSelectedPhase: […] When the victim sends its modified braids using the command send_modified, it builds a message and sends it to the attacker. The construction of the message is in the method com.cyberpointllc.stac.plaitme.command.DeliverModifiedPlaitCommand.executeEntity. The message contains the following information:

- the String generated by the above takePlaitString() method.
- the String representation of the braids 1,2,3,4 and 5

Statements on why the application does not contain a side channel (strength and reachability):

Even though the value of plaitIndex is used in takePlaitString, the expression this.plaitIndex / this.plaitIndex * plaitStr.length() is in fact equivalent to plaitStr.length(), which means that there is actually no leakage of the secret and the returned String is always the exact size of plaitStr.

How our tools supported this conclusion:

We used IST and the CodeSonar user interface to identify all the program locations that can generate network traffic. We analyzed each of them by manual browsing of the decompiled code to reach this solution.”

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.2.3 Question 029 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

The messages sent between two players are encrypted. The relevant code that performs this encryption is in the class com.cyberpointllc.stac.communications.fundamental.CommunicationsCryptoPhase: […]In order for the attacker to be able to view the plaintext messages, it has to guess the secret fields sessionCode and hmacCode of the class com.cyberpointllc.stac.communications.fundamental.CommunicationsCryptoPhase. These secrets are initialized when two players are connecting to each other, with a call to com.cyberpointllc.stac.communications.fundamental.CommunicationsCryptoPhase.fixSessionAndHmacCodes(BigInteger). This method is using a CodeExchanceServer that is initialized with a secret code obtained with the help of java.security.SecureRandom.SecureRandom() in com.cyberpointllc.stac.authorize.CodeExchangeServer.formRandomSecret().

Statements on any potential side channels within the application:

There would be a potential side channel if the time taken to setup a connection between two players would leak the secrets.

Statements on why the application does not contain a side channel (strength and reachability):

There does not seem to be a strong relationship between the observable and the secrets. The time to initialize a connection between two players, as well as the time to decrypt and encrypt messages, does not seem to be influenced by the private keys used by the cryptographic methods.

How our tools supported this conclusion:

We reached this conclusion by manual inspection of the decompiled code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"A member from the Colorado team pointed out the existence of a Chinese Remainder Theorem side channel attack on the RSA decryption, specifically MontgomeryTimeser.montgomeryMultiply […] The if-statement and associated mod operation causes a noticeable time difference on certain inputs, and Colorado showed empirical measurements of this difference. However, Colorado argued that an attack would require at least 60,000 operations, which exceeds the allowed budget of 40,000. Therefore, we believe this side channel is also insufficiently strong to leak the secret."

Live-Collaborative Evaluation:

GrammaTech was able to agree that a proposed attack by their collaborator would exceed the input budget and kept their correct answer from the Take-Home.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.3 BraidIt 4

A.6.2.2.3.1 Question 032 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"How the observables relate to the secret:

We understand the secret to be a number between 1 and 5, representing the number of the braid which was modified. This is stored as field weaveIndex of com.cyberpointllc.stac.weavethis.state.WeaveChosenState. The value of this field is set when the user runs the select_braid command. The observable is the size of network messages. The observable relates to the secret because the value of weaveIndex is used as a parameter to specify the width of the string representation of the modified braid. The relevant code snippet is as follows, found in com.cyberpointllc.stac.weavethis.state.WeaveChosenState: […] This method generates a String whose size is known from the value of this.weaveIndex * weaveStr.length(). When the victim sends its modified braids using the command send_modified, it builds a message and sends it to the attacker. The construction of the message is in the method com.cyberpointllc.stac.weavethis.command.TransferChangedWeaveCommand.execute. The message contains the following information:

- the String generated by the above obtainWeaveString() method.
- the String representation of the braids 1,2,3,4 and 5

Consequently, the observable (the packet size) allows the attacker to know the total length of these String values.

Why the observables reveal the value of the secret (even in the worst case) with a side channel that is sufficiently strong:

When receiving the above message, the attacker executes the method com.cyberpointllc.stac.weavethis.WeaveItDispatcher.handleReceivedMessageAid. This method builds the 5 initial braids and the modified braid given the information in the message, and creates a SelectionsState. However, when doing so, it shuffles the braids before the attacker can see them in the user interface with the command print. The executed code that performs the shuffling is the following, in com.cyberpointllc.stac.weavethis.state.SelectionsState […] This means that the incoming message has all the information to identify which braid is the modified one (because the other player does not shuffle the braids when building the packet), but the attacker has to be aware that the braids are shuffled after the message is received.

An easy way to avoid this problem is for the attacker to use a modified version of the program, that does not shuffle the braids: […]

Why this side channel is triggerable by an attacker Note, if the side channel is believed to be passively persistent, a statement to this effect is sufficient:

The attacker can just monitor the size of the incoming messages to guess which is the modified braid.

How our tools supported discovery of the vulnerability:

We used IST and the CodeSonar user interface to identify all the program locations that can generate network traffic. We analyzed each of them by manual browsing of the decompiled code to reach this solution."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. The index orders are shuffled on the benign user's side prior to sending and there are not enough oracle queries to map the leaked unshuffled index to the shuffled indexes received by the attacker.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In the collaborative discussion, we presented our attack and found that the Vanderbilt team had found the same one. However, we missed that there was a discrepancy in SelectionsState between the methods pullWeave and putWeave. There is an indirection to access weaves that is using the array weavePointers. This array is shuffled in the victim's program. During all the time the victim is choosing the braid and modifies it, the pullWeave method is actually not using the indirection. This is because isComplete is set to false. […] Just before sending the message with the generated braids and the modified braid, this isComplete field is flipped to true. Consequently, the weaveIndex field of weaveChosenState that is leaked from observing the size of the message does not actually correspond to the same index when the attacker receives the 5 unmodified braids. In other words, the victim's program has applied a shuffling to the 5 braids indexes that the attacker has to recover. However, after some discussions with the other teams, we have identified that the 5 auto-generated braids (generated by the victim) are using an unsecure pseudo-random generator java.util.Random. The method that generates the 5 braids is called 5 times and the relevant method is grabRandomWeave. […] First of all, if we are using a high number of strands, the second part of this method is unlikely to be executed: it is only executed if the generated braids is only composed of y's and z's. For the sake of simplicity in our justification, we assume it never happens (our attack would still work if this code is executed, but would introduce a little bit of noise). […] Weave.random is a random number generator based on java.util.Random, which means there is a way to recover the state of the random number generator from a sequence of its outputs, such that we can predict its next outputs. It is known that, for java.util.Random, it is sufficient to get two consecutive integers returned by random.nextInt() to recover the state. However, it is more complicated in our case, because the randomly generated integers are bounded, and the generator is also used to call nextBoolean.

Because the attacker receives the unordered set of 5 auto-generated braids, the purpose of the attack is to recover which one has been generated first, which one was second, etc. If we had that information, plus the fact that we have the side channel in the size of the message, it would be enough to have a working attack. The idea is to see, given one of the five braids, if we can guess what is the braid that was generated right after it. In java.util.Random, both the calls to nextInt(range) and nextBoolean() are calling the method next() that updates the state of the random number generator. To craft the attack, we used a tool called JavaCG (https://github.com/votadlos/JavaCG), that given a sequence of N integers generated by java.util.Random, is able to predict the result of the next calls. Because the complexity of this technique is much lower when the range argument of nextInt(range) is an even number, we used a range = 24, i.e. we used a game with 26 strands.

Unfortunately, in our case, we do not have the consecutive sequence of integer generated by nextInt(range) but only every other integers. This is because of the intermediate calls to nextBoolean(). Consequently, we had to modify the JavaCG program to handle the case where it does not have the entire sequence, but only a subset of it. Our experiments showed that it is not an issue and that JavaCG can still recover the state of the generator if the number of observations is large enough (a braid of size 15 is enough) Once we have that, we can try to feed to JavaCG each of the 5 braids, and JavaCG gives us the next 10 characters of the braid generated right after it. This is enough to recover the original order of the braids.

Here is an example when the attacker receives the following 5 braids:

(1) VgECJuFUCZqCvTKqmqchoVDuEk

(2) fOxqqgmEQgz

(3) JdMnTxMNpfZjNKnmHzfsxdGRoPEzydGErZuXtMKY

(4) GJtHyHpVmIFqlWcFWq

(5) JoSSLtLzfJcFuWYGMtZRvRlrwwfOdOJSxSlXDVXlwc

We ran JavaCG on each of these braids and it was able to guess the first 10 characters of the next braid:

$python attack.py

(1) VgECJuFUCZqCvTKqmqchoVDuEk -> foxqqgmeqg

(2) fOxqqgmEQgz -> jluvtpmfhf

(3) JdMnTxMNpfZjNKnmHzfsxdGRoPEzydGErZuXtMKY -> gjthyhpvmi

(4) GJtHyHpVmIFqlWcFWq -> jossltlzfj

(5) JoSSLtLzfJcFuWYGMtZRvRlrwwfOdOJSxSlXDVXlwc -> ivlyufxxrz

As we can see, we learn from that that (2) directly follows (1), and (5) follows (4) that follows (3).

Because the length of the braid (2) is a little too short, the result from JavaCG on (2) does not seem to match any braid. Similarly, the result of JavaCG on (5) does not match any other braid, which means that (5) is likely to be the last one. Just to make sure, we can re-run JavaCG by concatenating (1) and (2) to make it longer, and then it successfully identifies that (3) is the next one."

Live-Collaborative Evaluation:

We believe that GrammaTech's exploit to undo the shuffle procedure is feasible within the operational budget. However, as the java random library is out-of-scope, we have noted this as an out-of-scope vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.4 Calculator 1

A.6.2.2.4.1 Question 014 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

“Statements on any potentially vulnerable code structure in the application:

The application uses arbitrary-precision arithmetic code to perform mathematical calculations as requested by the attacker. The attacker can request that the application perform a multiplication of two very large numbers. When the numbers are above the relevant size threshold, the method GreatNumber.multiply selects the algorithm given in GreatNumber.transformMultiply to perform this multiplication. That algorithm uses the Transform.primaryTransform method to perform a Fourier transform on the coefficients of the arbitrary-precision integers to be multiplied. That method allocates several large arrays of Complex numbers. In some cases, the memory used by these arrays is very large.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures:

We empirically investigated the memory usage of the multiplication code by constructing a variety of different inputs. One of the inputs that produced consistent high memory usage was a sum of several repetitions of the following expression, submitted to the Roman numeral calculator: $(M^M * M^M * M^M * M^M * M^M * M^M * M^M * M^M) * (M^M * M^M * M^M * M^M * M^M * M^M * M^M * M^M) - (M^M * M^M * M^M * M^M * M^M * M^M * M^M * M^M) * (M^M * M^M * M^M * M^M * M^M * M^M * M^M * M^M)$

We chose this expression because it passes through several size checks in the multiplication code that prevent excessively large numbers from being computed. Note that $M^M = 103000$, so the subexpression $(M^M * M^M * M^M * M^M * M^M * M^M * M^M * M^M) = 1024000$, so the product $(M^M * M^M * M^M * M^M * M^M * M^M * M^M * M^M) * (M^M * M^M * M^M * M^M * M^M * M^M * M^M * M^M) = 1048000$ is very close to the limit of 1050000 imposed by the code.

Statements on why the application does not contain a complexity vulnerability (strength and reachability):

We measured the memory used by the calculation described above using a profiler and the pmap command. Using the profiler, we found that the large arrays of Complex numbers were responsible for the large memory usage of this calculation. We found that memory usage sometimes went over the threshold of 4GB given in the challenge problem question. However, the memory usage was non-deterministic, presumably depending on the behavior of the garbage collector, and most of the times that we submitted this input to the calculator, the memory usage did not cross the 4GB threshold. The challenge problem question specifies that an attack should cross that threshold with a probability of 80%, and our empirical measurements showed that this threshold was crossed with far lower probability. For this reason, we decided to answer this question in the negative.

How our tools supported this conclusion:

Our Inspector Space Time (IST) tool has heuristics that find hotspots (which we call "kernels") in the code that are predicted to have high resource usage. The kernel that IST's heuristics ranked most highly for predicted memory usage included the transformMultiply method along with several other methods of the GreatNumber class. The kernel that was ranked second highest consisted of the transformMultiply method and the six main methods of the Transform class.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability; however, Colorado and Northeastern identified unintended vulnerabilities in the challenge program.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“We discussed this problem with several teams including Northeastern and Iowa.

Northeastern attacked the code of divide in GreatNumber.

This code is vulnerable because it allocates many arrays during the course of performing a division. (Northeastern used the same attack for the memory ACV question related to calculator_1 also.) Iowa presented an attack on the multiplication code which was similar to many of the attacks that we attempted during the take-home portion of the engagement. (In fact, an attack that we developed during the take-home portion would have crossed the memory-usage threshold if we had tested it on the NUC; unfortunately, during the take-home portion, we tested memory usage only on a virtual machine.) The attack presented by Iowa targeted transformMultiply and the primaryTransform method of the Transform class.

The divide method makes many calls to the append method, which will result in an allocation of a large array. The primaryTransform method also allocates many large arrays. Multiple teams reported that memory usage was higher on the NUC, and they attributed the garbage collection behavior of that platform, which is much less aggressive and therefore vulnerable to high memory usage. The Northeastern team presented an attack that computes a large number such as 5*10^499 and then repeatedly divides it by 1, 400 times. This results in many divisions being calculated without decreasing the size of the number. The Iowa team used the expression (9^5247) (multiplied by itself 5 times), multiplied by (9^1048) (multiplied by itself 25 times), multiplied by 9^1010. This generates a number having 50,000 digits, which is the maximum number allowed. Iowa sent this same query 62 times in a row, with a 0.1 second delay between the different queries. While trying to answer 62 copies of this query simultaneously, the memory usage exceeds the 4GB threshold. We empirically verified that the memory usage exceeded the threshold when we performed Northeastern’s attack.

Other teams discovered this vulnerability using a mix of fuzzing and manual investigation.

Live-Collaborative Evaluation:

Following the collaboration, GrammaTech was convinced of the other teams’ findings and changed their answer correctly.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.4.2 Question 016 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“Code that contains vulnerability:

The application uses arbitrary-precision arithmetic code to perform mathematical calculations as requested by the attacker. The attacker can request that the application perform a multiplication of two very large numbers. When the numbers are above the relevant size threshold, the method GreatNumber.multiply selects the algorithm given in GreatNumber.transformMultiply to perform this multiplication. That algorithm uses the Transform.primaryTransform method to perform a Fourier transform on the coefficients of the arbitrary-precision integers to be

multiplied. This method performs a large number of recursive calls and works with large arrays of Complex numbers. For these reasons, the attacker can cause the program to run for a long time.

Why the code contains a vulnerability:

The code performs many size checks on various parts of input expressions that prevent excessively large numbers from being computed; these checks prevent many expressions from causing excessive resource usage. However, an expression that is constructed by multiplying 105000 by itself several times can pass the checks and yet cause long running time.

Why the vulnerability is exploitable:

We found that the expression printed by the following Python one-liner, when submitted to the basic calculator, consistently resulted in a long running time: `print "+".join(["((10^5000)*(10^5000)*(10^5000)*(10^5000)*(10^5000))"]*194)` This expression is less than 10 KB in length, which is important because the calculator_1 program checks the sizes of input expressions and rejects those that are over 10 KB. Processing of this expression takes about 100 seconds, which is less than the threshold of 120 seconds specified in the problem. However, if two copies of the expression are sent to the calculator concurrently, then a concurrent benign request to the calculator is slowed down by 200 seconds or more, which is well above the threshold. These requests amount to a total of 23 KB of network traffic sent by the attacker, which is well under the input budget of 60 KB specified in the challenge problem question. For these reasons, we have answered the question in the affirmative.

How our tools supported discovery of the vulnerability:

Our Inspector Space Time (IST) tool has heuristics that find hotspots (which we call "kernels") in the code that are predicted to have high resource usage. The kernel that IST's heuristics ranked most highly for predicted running time included the transformMultiply method along with several other methods of the GreatNumber class. The kernel that was ranked third highest consisted of the transformMultiply method and the six main methods of the Transform class."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The unintended vulnerability reported by GrammaTech was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.5 Calculator 2

A.6.2.2.5.1 Question 003 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Statements on any potentially vulnerable code structure in the application:

[…] The method AbstractHttpCoach.handle writes exception tracebacks to a log file when an exception of type ExecutionException or InterruptedException occurs during a call to Future.get(). The particular Future object on which get is called is a wrapper that wraps most of the HTTP-request-handling code of the application, including the code that performs the mathematical calculations requested by the user. An uncaught exception that occurs during the execution of the Future will result in an ExecutionException. Thus, one way to cause data to be written to the log file is to send a mathematical expression to the calculator that causes an exception to be thrown and ultimately caught in AbstractHttpCoach.handle. Knowing this, we investigated the question of what exceptions can be thrown by the calculation-handling code, and we eventually focused on the Mather.calculateOutcome method.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures:

The Mather.calculateOutcome method is potentially vulnerable because it performs a recursive call to handle any parenthesized sub-expression of the expression that it is analyzing. Thus, when attacker supplies an expression having deeply nested parentheses, calculateOutcome will perform deep recursion; at a sufficiently great depth, a stack overflow exception is thrown that results in an ExecutionException, which causes a traceback to be written to the log file. Because the recursion is deep, the traceback itself is very long.

Statements on why the application does not contain a complexity vulnerability (strength and reachability)

We empirically measured the number of parentheses required to cause a stack overflow in calculateOutcome. The Roman numeral expression M^M, surrounded by 9850 pairs of parentheses, reliably causes a stack overflow that writes a 77 KB traceback to the log file. It is possible to send more than one expression, sequentially, to the server. However, because this expression is approximately 20 KB, the input budget of 60 KB prevents us from sending a large number of copies of it to the server. Because this question's file size threshold is 4 MB, approximately 52 copies of the aforementioned expression would have to be sent to the server to cross the threshold, but this goes far beyond the question's input budget. For this reason, we answered this question in the negative. (If we had found some means of compressing this input or amplifying the size of the corresponding log file output, we would have answered the question in the affirmative.)

How our tools supported this conclusion:

We ran fuzzing experiments using a fuzzing script that we have recently started developing. In one experiment, our fuzzer constructed many mathematical expressions that are generated by a simple grammar for mathematical expressions. In another experiment, our fuzzer constructed random strings. For both experiments, our goal was to find expressions that, when sent to the calculator, would throw exceptions that would cause tracebacks to be written to the log file. These experiments were not successful in finding such exceptions. A manual investigation revealed the possibility of a stack overflow caused by an expression with very deeply nested parentheses.”

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.5.2 Question 005 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“Code that contains vulnerability:

The application uses arbitrary-precision arithmetic code to perform mathematical calculations as requested by the attacker. The attacker can request that the application perform a multiplication of two very large numbers. When the numbers are below a particular size threshold, the method BigInteger.multiply selects the algorithm given in BigInteger.plainLogspaceMultiply to perform this multiplication.

Why the code contains a vulnerability:

The method BigInteger.plainLogspaceMultiply contains code that, when decompiled, looks like this: […] The large number of loops in this method, including the inner while loops that serve to randomly increase running time, cause the method to be relatively expensive. An attacker can cause this method to be executed by requesting many multiplications of numbers in which at least one number is at most 101000, because when that criterion is met, the multiply method of BigInteger calls plainLogspaceMultiply in preference to reformulateMultiply.

Why the vulnerability is exploitable:

We sent the following expression as input to the basic calculator. Note that the expression contains newline characters for the purposes of presentation only; so, if you wish to send this expression to the calculator, please remove all newlines first. […] Sending this request delayed the server's response to the supplied benign request script by 8 minutes, which is well over the 120 second threshold specified in the question. This attack requires sending less than 4 KB of traffic to the server, which is well within the budget of 60 KB for this question. For these reasons, we have answered the question in the affirmative.

How our tools supported discovery of the vulnerability:

Our Inspector SpaceTime (IST) tool has heuristics that find hotspots (which we call "kernels") in the code that are predicted to have high resource usage. The kernel that IST's heuristics ranked most highly for predicted running time included the plainLogspaceMultiply method of the BigInteger class, along with several other methods of BigInteger.”

Take-Home Evaluation:

The challenge contains an intended vulnerability in parsing input expressions. GrammaTech reported an unintended vulnerability that was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly identified and unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.5.3 Question 009 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Statements on any potentially vulnerable code structure in the application:

The application uses arbitrary-precision arithmetic code to perform mathematical calculations as requested by the attacker. The attacker can request that the application perform a multiplication of two very large numbers. When the numbers are above the relevant size threshold, the method BigInteger.multiply selects the algorithm given in BigInteger.reformulateMultiply to perform this multiplication. That algorithm uses the Reformulate.chiefReformulate method to perform a Fourier transform on the coefficients of the arbitrary-precision integers to be multiplied. That method allocates some large arrays of Complex numbers. The alternative multiplication algorithm is implemented by BigInteger.plainLogspaceMultiply.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures:

Our investigations focused mostly on the use of memory by the Reformulate.chiefReformulate method. Like the corresponding method in the calculator_1 problem (i.e., Transform.primaryTransform), the chiefReformulate exhibits multiple recursion. However, chiefReformulate calls itself at most 5 times, whereas primaryTransform calls itself at most 33 times. Because of this design difference, each call to chiefReformulate also allocates fewer large arrays of Complex numbers than primaryTransform does. For this reason, chiefReformulate has much lower memory usage than primaryTransform.

Statements on why the application does not contain a complexity vulnerability (strength and reachability):

We empirically tested the memory usage of this program by constructing expressions containing multiplications of large numbers. As with the other calculator problems, we tried using expressions containing many multiplications of 10^{1000}, and 10^{5000}, and M^M, among other expressions. Our testing indicated that the memory used by our chosen expressions was far below the 4GB threshold given in the question.

How our tools supported this conclusion:

Our Inspector SpaceTime (IST) tool has heuristics that find hotspots (which we call "kernels") in the code that are predicted to have high resource usage. The kernel that IST's heuristics ranked most highly for predicted memory usage consisted of the methods of BigInteger and one other method. The kernel that ranked second highest for predicted memory usage consisted of the methods of the Reformulate class, including chiefReformulate, along with BigInteger.reformulateMultiply."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability; however, Northeastern and Two Six Labs identified an unintended vulnerability in the challenge program.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“We discussed this problem with several teams including Northeastern and Iowa. Northeastern attacked the code of divide in BigInteger. This code is vulnerable because it allocates many arrays during the course of performing a division. (Northeastern used the same attack for the memory ACV question related to calculator_1 also.) Iowa presented an attack on the multiplication code which was similar to many of the attacks that we attempted during the take-home portion of the engagement. (In fact, an attack that we developed during the take-home portion would have crossed the memory-usage threshold if we had tested it on the NUC; unfortunately, during the take-home portion, we tested memory usage only on a virtual machine.) The attack presented by Iowa targeted plainLogspaceMultiply method of the Reformulate class.

The divide method makes many calls to the append method, which will result in an allocation of a large array. The plainLogspaceMultiply also allocates a large array when given a large number as input. Multiple teams reported that memory usage was higher on the NUC, and they attributed the garbage collection behavior of that platform, which is much less aggressive and therefore vulnerable to high memory usage.

The Northeastern team presented an attack that computes a large number such as 10^499 and then repeatedly divides it by 1. This results in many divisions being calculated without decreasing the size of the number. The Iowa team created an expression that multiplied two numbers. The first number consists of 48,000 copies of the digit "9". The second number consists of 1001 copies of the digit "9". Because the degree of a number is the number of digits minus

1, the second number has a degree of 1000. Thus, the multiply method compares the degree with 1000 and finds the degree to be lower; thus, it selects the plainLogspaceMultiply algorithm to perform this multiplication. We empirically verified that the memory usage exceeded the threshold when we performed Northeastern’s attack and also when we performed Iowa’s attack.

Other teams discovered this vulnerability using a mix of fuzzing and manual investigation.

Live-Collaborative Evaluation:

Following collaboration, GrammaTech was convinced by other teams’ findings to change their answer correctly.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.6 CaseDB

A.6.2.2.6.1 Question 019 (SC Space, Intended Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Statements on if and how the observables relate to the secret:

When a privileged user makes a GET request the server will send the documents to the user at port 6166 each with a new response. Then the nodes of the server will send the contents of the documents to the user at port 6167. When the server sends the documents to port 6166, the number of responses will correspond to the number of documents returned by the GET request.

An attacker can determine the number of public documents associated with a case. Thus, the number of responses will reveal if a request contained an informant or not. Regarding the identity of an informant, we first note that the contents of the documents, that is the responses to port 6167, do not necessarily contain the identity of an informant. These files are also encrypted. In other words, the responses to port 6167 do not relate to the informant's identity. When the server sends documents to port 6166, the server redacts the identity of confidential informants, by replacing the characters of the name of the informant with x's. This means that even a privileged user does not see the identity of confidential informants. The number of x's in the response corresponds to the number of characters of the informant's name. Thus, the observable of the packets sent to port 6166 do leak the information of the length of the name of the informant. It is also possible for a secret document to be related to the plan which is view-able to the attacker. If an unprivileged attacker makes a unique request to the server, the documents retrieved by the server will not contain any secret documents because the attacker is unprivileged. However, suppose a privileged user makes a GET request, and then the attacker makes an identical GET request or an equivalent SEARCH request. Because the server uses a query caching mechanism, the query plan in CaseDBDocController.runDocQuery will contain the secret document from the privileged user's GET request. This plan will get sent to the nodes. Therefore, in this situation there is a relation between secret documents and the plan.

Statements on any potential side channels within the application:

An attacker can identify if a GET response contains an informant. This is accomplished by first identifying the case associated with the privileged request. The case name is in plain-text. The attacker also notes the number of responses sent to the privileged user's port 6166. This is the number of docs sent to the privileged user. The unprivileged attacker then makes the same GET request and notes the number of returned public documents. If the number of documents differ, then the first GET response contained an informant.

Statements on why the application does not contain a side channel (strength and reachability):

We do not believe it is possible to determine the identity of an informant. Primarily because the identity of the informant is redacted even when it is sent to a privileged user.

We also do not believe it is possible to exploit the relation between secret documents and the plan, under the circumstance when a privileged user has made a GET request. To exploit this leakage, the attacker would need to be able to make SEARCH or GET queries which would affect the plan sent from the server to the nodes, based on the value an informant's identity. We have not observed any situation where a particular SEARCH or GET query has affected the plan sent from the server to the nodes, which are affected by the value of the secret.

How our tools supported this conclusion:

We reached this conclusion through manual inspection of the decompilation."

Take-Home Evaluation:

GrammaTech did not identify the intended vulnerability. The team correctly identified the caching behavior that could allow an unprivileged attacker to "touch" restricted documents; however, the team missed the vulnerability that allows an attacker to sequentially determine the informant in restricted documents.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Though GrammaTech and other teams collaborated on CaseDB for ~10 hours during the live engagement, they spent little time answering the actually vulnerable question, focusing instead on Question 36.

Post-Engagement Analysis Ruling: Incorrect

A.6.2.2.6.2 Question 021 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"[…] Statements on any potential side channels within the application:

An attacker can identify if a GET response contains an informant. This is accomplished by first identifying the case associated with the privileged request. The case name is in plain-text. The attacker also notes the how long it took for the server to send responses to the privileged user's port 6166. This is the number of docs sent to the privileged user. The unprivileged attacker then makes the same GET request and notes the amount of time the response to its own port 6166 took. If the times differ, then the first GET response had more documents in it and thus contained an informant.

Statements on why the application does not contain a side channel (strength and reachability):

We do not believe it is possible to determine the identity of an informant. Primarily because the identity of the informant is redacted even when it is sent to a privileged user. We also do not believe it is possible to exploit the relation between secret documents and the plan, under the circumstance when a privileged user has made a GET request. To exploit this leakage, the attacker would need to be able to make SEARCH or GET queries which would affect the plan sent from the server to the nodes, based on the value an informant's identity. We have not observed any situation where a particular SEARCH or GET query has affected the plan sent from the server to the nodes, which are affected by the value of the secret.

How our tools supported this conclusion:

We reached this conclusion through manual inspection of the decompilation."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.6.3 Question 036 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“[…] for a user to view a secret document

1) The user needs to have the secret document sent in the request to a node.

2) The "user" header field needs to have been populated by the server.

3) The user needs to have the "key" associated with the "user" value in order to decrypt the contents.

Statements on if and how the observables relate to the secret

As stated above the attacker needs 3 things to happen in order to view secret documents. First, the attacker needs to make a request to the server which results in a secret document being sent to a node. This is possible. If an unprivileged attacker makes a unique request to the server, the documents retrieved by the server will not contain any secret documents because the attacker is unprivileged. However, suppose a privileged user makes a GET request, and then the attacker makes an identical GET request or an equivalent SEARCH request. Because the server uses a query caching mechanism, the query plan in CaseDBDocController.runDocQuery will contain the secret document from the privileged user's GET request. This plan will get sent to the nodes. Thus, the attacker can use the observable of a privileged GET request to determine an identical GET request or equivalent SEARCH request to ensure a secret document is sent to a node. Second, the attacker needs to ensure that the "user" header field from the exchange between the server and the node is set. The only place the "user" header field is populated in in GetCase.send. In this method the server decrypts the incoming password using the user's saved key and the incoming initialization vector. Upon successful decryption, the server will check if the user is privileged or not. If the user is privileged the "user" header will be set and a getCaseDocs method will be called. If the user is not privileged the "user" header will not be set. In other words, for an attacker to make sure the "user" header field is set in the exchange between the server and a node, she will need to make a GET request as a privileged user. When the privileged user makes a GET request, the privileged user's password is encrypted using their corresponding key as well as a given initialization vector. The attacker can observe the traffic between the privileged user and the server to see the privileged user's encrypted password, as well as the user initialization vector. Finally, if the attacker can get a node to send over an encrypted secret document, the attacker needs to make sure she can decrypt the sent file contents. This means the attacker needs to know the key used for encryption. The attacker directly knows her own key. The attacker also can observe the privileged user's encrypted password and used initialization vector in a GET request. The encrypted password and initialization vector is related to the privileged user's key in that that password was encrypted with the key and initialization vector.

Statements on any potential side channels within the application:

Consider the case when a privileged user makes a GET request. One peculiar side effect we noticed is that NodeSvr.getItem throws an exception when an unprivileged user makes a SEARCH request that is equivalent to the privileged GET request, but an exception is not thrown when an unprivileged user makes a GET request identical to the privileged GET request. We know this occurs because GetCase.send sets the exchange header "key" before a check is made to see if the user is privileged or not; however, SearchDoc.send never sets the exchange header "key". This means the NodeSvr.getItem will throw an exception when trying to decode the "key", when an unprivileged user makes a SEARCH request but not a GET request. While we recognize this as peculiar behavior, we have no way of exploiting it.

Statements on why the application does not contain a side channel (strength and reachability):

As stated previously, three things need to happen for an attacker to view a secret document. We do not believe an attacker can view a secret document, because we do not see how (2) and (3) can happen at the same time. To see why, suppose that by some means the attacker obtains information that allows her to sign in as a privileged user. This will succeed in getting a node to send over a file containing a secret document's contents. However, this file will be encrypted using the privileged user's key, which is unknown to the attacker. On the other hand, suppose, the attacker has gotten a node to encrypt a secret document using the attacker's key. However, this is impossible, because in this case the attacker will not have signed in as a privileged user. This means the server will not set the "user" header in the exchange between the server and the nodes. Therefore, in this case no secret documents will ever be sent, encrypted or not.

How our tools supported this conclusion:

We reached this conclusion through manual inspection of the decompilation."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Our understanding of the meaning of this question has changed. We now interpret that the question asks if there exists an exploit which allows an unprivileged attacker to see an encrypted version of a secret file.

For this question the attacker is attempting to receive encrypted secret files. The file contents are sent by the nodes. If a file is a secret file the nodes then encrypt the file with a corresponding secret key. It is possible for secret files to be in the plan that is sent to the nodes. However, during normal operation the nodes filter out secret files when responding to an unprivileged user in the following way. In NodeSvr.getItem, a node is sent a series of documents to relay to the client. In NodeSvr.getItem the nodes iterate over all the given documents. They then determine if each document is a secure document or not. If a document is a secure document the node will then access the user and key header in the exchange which is provided as an argument to NodeSvr.getItem. During normal operation, the presence of these headers determine whether a secure file is sent or not. That is, if the server does not set the user and key headers for the exchange provided to NodeSvr.getItem then the nodes will not send secure documents. Thus, in order to get secure file contents to be sent from node to the attacker, the attacker needs to make sure that the server sets the user and key header fields.

The only place in the application where the server might set the user header is in the GetCase.send method. Specifically, a user provides a user name, encrypted password, and initialization vector to the exchange which is provided to the GetCase.send method. The server then internally looks up the AES key associated with the given username and decrypts the given encrypted password using the given initialization vector and the stored AES key to obtain the plaintext password. The server then compares the decrypted password with the one locally stored on the server. If the passwords match, the server checks the value of u.isPriv. It is only then, that the server will set the user header for the exchange given to NodeSvr.getItem.

With all this in mind, we interpret that the only way to receive secret file contents is to authenticate as a privileged user.

The server authenticates a user in a potentially unsafe way. The server decrypts passwords using an AES util, which uses CBC mode. What this means is that the server will decrypt the first block of a ciphertext as follows. First, the server will pass the ciphertext through AES using an internal secret key. Then the initialization vector is XOR'd with the result. The result of the XOR is the first block of the decryption result. This application is therefore potentially vulnerable because passwords are no longer then 8 characters long. This means the plaintext (and ciphertext) will always consist of one block. What this means is that for any given ciphertext there will exist an initialization vector, which can cause the result of the decryption to be any arbitrary value, namely the plaintext password of a privileged user.

There is a potential side-channel in the application which would allow an attacker to find such an initialization vector. When the server decrypts a given encrypted password using a given initialization vector and a stored secret key, the server checks the resulting plaintext against a stored version, which matches characters between the two strings. Also, the server throws an exception each time a character mismatches. If an attacker could measure the execution time of the server to detect the number of mismatched characters an attacker could make a series of GET requests with the same encrypted password, but different initialization vectors, and determine the number of matches. The attacker can continue modifying the initialization vector bytes until more and more matches occur, until a successful login.

We do not believe the attack described above is exploitable for two reasons. The first is that we have empirically measured response times with varying matched characters and we have not observed a significant time difference to determine the number of matches between the stored plaintext and the decrypted plaintext. Also, we cannot average out any of these timing variances, because the application does not permit initialization vectors to be reused. Thus, for each initialization vector we can make only one measurement.

We reached this conclusion through manual inspection of the decompilation as well as discussion with the Northeastern and Vanderbilt teams."

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable. The team correctly noted the clarified interpretation of the secret to be leaked an noted the potential avenues of attack that we observed in our own analysis of the challenge.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.7 Chessmaster

A.6.2.2.7.1 Question 006 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

The communication between the victim to the server encrypts packets. However, the packets sent from the victim to the server contain the password of the victim, and the packets from the server to the victim will contain the game state. Both of these pieces of information can be thought of as

secrets. Their encrypted versions are contained in the view-able communication between the victim and the server.

Statements on any potential side channels within the application:

We identify three potential side channel attacks within the application.

1) An attacker could determine the password of the victim.
2) An attacker could cause the server to inadvertently send the game state of a different user to the attacker.
3) An attacker could examine the network traffic of the benign user interacting with the application, and directly determine the game state from this information.

Statements on why the application does not contain a side channel (strength and reachability):

We argue for why none of the above side-channels are sufficient to determine the game state of a benign user.

1) (Determining the password of the victim) The packets from the benign user to the server contain the password; however, these packets are encrypted. Thus, we do not believe it is possible to determine the password by viewing the benign user's traffic. Alternatively the attacker could try and determine the password by via active operations. That is, the attacker would examine the response time of CheckLogin in Server.java on the username of the victim and some attacker chosen password string. However, we do not believe this strategy will reveal any information about the password. CheckLogin uses String.equals to compare passwords, which doesn't appear to leak enough information about its operands to warrant an attack.
2) (Server reveals the game state) The server services requests via the method serve in Server.java. The response sent from the server is stored in the static variable Server.response. Server.response is set to the empty string at the beginning of the serve method. Consider when the victim makes a move request. In this case the serve method will populate Server.response with the state of the game, along with some other information. Since Server.response is static, the Server.response variable for any other user will contain a string representation of the game state for the victim. If the attacker could get the server to send Server.response when Server.response contains the victims gain state the attacker would then be able to obtain the secret. We do not believe this is possible. Since Server.response is set to the empty string at the beginning of serve, some thread interleaving would be required to send the victim's state to the attacker. Unfortunately, for the attacker the serve method is synchronized, so it appears that only one thread can be in the serve method at a time. Therefore, the required thread interleaving does not seem to be possible.
3) (Gain game state information through passive operations) During a game with a benign user encrypted packets are sent from the server to the victim which contain the game state. It might be possible for the attacker to use the timing or size information of these packets to narrow the set of possible game states. However, we do not believe this is sufficient. The game state of chess is too large. Thus, it does not seem possible that any passive operations will be able to narrow the set of possible games to a single game state.

How our tools supported our conclusion:

We arrived at our conclusion through manual inspection of the decompiled code.”

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“We would like to add to our response why we believe it is not possible to gain game state information through passive operations. There is definitely information that is leaked with time. The chess AI does take different amounts of time to calculate next moves depending on the current state of the game. However, we do not have any qualitative way to exploit this. That is while it is true that the chess AI takes different amounts of time, we currently have no way to know what states these times correspond to.”

Live-Collaborative Evaluation:

The point difference calculated by the AI is the primary contributor of the move runtime; however noting the potential equivalent states and moves that can result in a given point difference along with the potential for cycles in movements resulting in states that are computationally equivalent in processing time, we agree with the team’s conclusion that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.7.2 Question 015 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“Code that contains vulnerability:

In State.java pieces are defined point values, which are used to heuristically evaluate a board and determine who is winning. We will say the score is the sum of the values of white pieces on the board minus the sum of the values of black pieces on the board. The AI in GameClient.java performs a negamax search over all moves of a certain depth. This depth is determined by the following expression 3 - (4.1*10-4)x + (3.9*10-8)x2, where x is the current score of the board. What this means is that if the score is either very positive or very negative, the depth will increase and the server will take a long time to determine the next move. Under normal chess circumstances x cannot be large enough (or small enough) to result in a depth that can answer the question. Like in real chess pawns can be promoted to Q, R, N, or B if they make it across the board. A pawn cannot be promoted to a king. The application prevents the user promoting a pawn to a king by making the check, lines[4].toLowerCase().equals("king"), in Server.java. This check is not sufficient and it is possible to get the application to promote a pawn to a king. In this situation the score of the board will be very lopsided towards white, and thus the search depth for negamax will sufficiently increase to effectively shutdown the server.

Why the code contains a vulnerability:

Server.java contains a vulnerability, because lines[4].toLowerCase().equals("king") is not sufficient to prevent a pawn being promoted to a king. Suppose lines[4] = kng. In this case lines[4].toLowerCase().equals("king") will be false. In this case the application will then call lines[4] = this.corrector.correct(lines[4]).toUpperCase();. Since, kng is closest to king, the autocorrecter will modify lines[4] to be KING after lines[4] = this.corrector.correct(lines[4]).toUpperCase(); is called. Thus, by misspelling king the attacker

has bypassed the check to make sure a pawn cannot be promoted to a king. Once, this check has been bypassed the score will be lopsided towards white. […]"

Take-Home Evaluation:

GrammaTech identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.8 Class Scheduler

A.6.2.2.8.1 Question 012 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

The observable is the time between the packets sent and received through the network. The secrets are the size of two fields private HashMap<Integer, Teacher> teachers and private HashMap<Integer, Student> students of the class com.bbn.classScheduler.config.SchoolData. The two fields are set to the uploaded school data in the method SchoolData.XMLReaderDOM. The processing time of this method is linear in the number of teachers, students, courses, and rooms. Also, the method createSchedule in the class com.bbn.classScheduler.scheduler.GenerateScheduleThread invokes a genetic algorithm to generate a class schedule with respect to the uploaded school data. During this process, the running time of the method getRandomStudents of the class com.bbn.classScheduler.objects.Population dependent on the number of students only if these are less than getMaxStudents() (which is 20): […]

Statements on any potential side channels within the application:

There could be an SCL in time in the following situations:

- the class schedule generation time depends on the number of teachers and students, or
- the authentication of the application contains an SCL in time.

Additionally, when a new user starts a session (even before the log in) a new vaadin UI is created and the method vaadinUI.init is called. This method loads the existing school information, so its execution time depends on the number of students and teachers.

Statements on why the application does not contain a side channel (strength and reachability):

Suppose the genetic algorithm reaches the maximal number of generations. Then the number of teachers barely influences the generation time, and the number of students could affect the running time through the Population.getRandomStudents methods. However, even just for the later case, the application samples some students randomly, hence it is unlikely to infer the number of students from the running time. The method authenticate of the class com.bbn.classScheduler.scheduler.Authentication makes use of the method String.equals to

check credentials. This method does not have a constant time. However, because the operation budget is only 40, it is unlikely to obtain the password through side-channels. We performed experiments by starting sessions after processing different school files. We tried to observe if there is a noticeable difference in timing depending on the number of students or teacher. We did not observe a significant difference. Even if a difference was noticeable, we would have no way to attribute those differences to a number of students or teachers.

How our tools supported this conclusion:

We proceeded by manually inspecting the decompiled source code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"During the live engagement Utah shared that the progress bar endpoint is not protected by the session. That means we can know the progress of a schedule generation without user privileges. We explored that possibility further and Iowa made measurements to try to relate the timing and the progress. However, the progress of class scheduler seems to be linear and largely independent on the number of students and teachers. We found no way to obtain further information."

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.8.2 Question 017 (AC Time, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Statements on any potentially vulnerable code structure in the application:

The only time-consuming request is the class schedule generation, i.e., the method createSchedule in the class com.bbn.classScheduler.scheduler.GenerateScheduleThread. There would be a complexity vulnerability if this request could block other benign requests for a long time.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures:

The application uses a genetic algorithm to generate class schedule. The algorithm proceeds by initializing a random collection of candidate schedules, and iterating for at most maxGenerations times to perform crossovers and mutations. The time complexity of the method com.bbn.classScheduler.objects.Population.initialize is about $O(populationSize * numberOfCourses * numberOfStudents)$, where populationSize is the size of the initial collection, and the algorithm needs to iterate all the courses and assigns some students to each course. The time complexity of the method com.bbn.classScheduler.objects.Population.generateNextGeneration is in $O(selectionNumber * selectionPoolSize + selectionNumber * numberOfCourses + selectionNumber * mutationSelectionRate * numberOfCourses)$, where selectionNumber

is the number of candidate schedules selected in each iteration by randomly sampling selectionPoolSize candidates and choosing the fittest one. The algorithm first generates a selection pool, and then the crossover and mutation process take in the order of $O(numberOfCourses)$ and $O(mutationSelectionRate * numberOfCourses)$ amount of time per candidate, respectively.

Statements on why the application does not contain a complexity vulnerability (strength and reachability):

The Spring framework used to implement the application makes use of a non-blocking IO connector. That means that application would return a blank file to a request for the most recent schedule if the server does not have an existing schedule and it is generating a new one. Therefore, benign requests would not be delayed. Furthermore, the running time of the class schedule generation itself would not exceed 50 minutes under the default system configuration. On the other hand, if the application is configured in the following way, the time of schedule generation could exceed the budget."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Colorado found a way to make the schedule generation take a very long time by adding courses with negative number of sections. This allows adding other courses with very high number of sections (only the total is checked) which in turn makes the schedule generation take a very long time. There is still a problem, our argument about the web framework being non-blocking still applies. However, we made some measurements and if the schedule generation is triggered a sufficient number of times (in our experiments 5 is enough), the whole application becomes unresponsive. Note that it is possible to start multiple schedule generations because the field boolean generationInProcess of VaadinUI is not static and each session contains a different instance. The attack is as follows. First, we login, upload school data with courses containing a huge amount of sections and start the schedule generation. Then, we login and start generating the schedule 4 more times (there is no need to upload a new school data file). At that point the application stops responding all requests."

Live-Collaborative Evaluation:

Through collaboration GrammaTech identified an unintended vulnerability. We believe that the issue flagged by the team is present in that multiple schedule generations of sufficient complexity can be triggered at the same time causing the server to stop processing requests – a valid DoS vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.8.3 Question 028 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

The observable is the time between the packets sent and received through the network. The secret is the field private Schedule finalSchedule of the class

com.bbn.classScheduler.scheduler.VaadinUI. The class schedule of a teacher is retrieved in the method com.bbn.classScheduler.objects.Schedule.getMyClasses(Teacher). This method iterates all the courses and checks if each course is taught by the given teacher. There is a positive correlation between the method's running time and the number of courses taught by the given teacher because of the following non-constant-time branch statement. […]

Statements on any potential side channels within the application

Because in this question, no benign users are interacting with the application, there can be an SCL in time if and only if the authentication of the application contains an SCL in time.

Statements on why the application does not contain a side channel (strength and reachability)

The method authenticate of the class com.bbn.classScheduler.scheduler.Authentication makes use of the method String.equals to check credentials. This string comparison method of JDK is not constant-time. Therefore, there seems to be a side channel in time that could be used to obtain the password. However, and even though the operation budget is 30000, the timing difference caused by string comparison is too small relative to the timings of the network traffic. We performed several experiments and were unable to obtain any useful information from the timing of the login attempts. Therefore, we conclude that the side channel is not exploitable.

How our tools supported this conclusion:

We proceeded by manually inspecting the decompiled source code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.9 Effects Hero

A.6.2.2.9.1 Question 010 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

The observable is the time between the packets sent and received through the network. The secret is the field private ByteArrayOutputStream memory of the class com.bbn.hero.FileOutputBlock. This field is computed in the method FileOutputBlock.process that is invoked by the method VaadinUI.runStage (in file com.bbn.hero.VaadinUI, line 155), which runs several processing blocks on a user-input sound file. The application might take some time to generate the secret sound file.

Statements on any potential side channels within the application:

We can think of three ways in which there could be an SCL in time in this application:

- if, when setting up the sequence of blocks in the stage, the interaction of the victim with the VaadinUI creates network traffic that allows the attacker to guess the exact sequence of blocks and inputs.
- if, when running the stage, i.e. when executing the VaadinUI.runStage method, the network traffic is such that the attacker can identify the exact sequence of blocks and inputs.
- if the network traffic leaks the final output of the stage.

Statements on why the application does not contain a side channel (strength and reachability):

We believe that, for each of the three above points, there is no exploitable side channel:

- the victim uses the web interface interactively, which means that the timestamp of its outgoing requests is irrelevant. What could still be relevant is the response time of the server after each outgoing requests (e.g. when adding/deleting a block, or adjusting the settings/inputs of an existing block). However, just looking at the server's response time is clearly not sufficient to guess the exact sequence of actions of the victim.
- when running the sequence of blocks, i.e. the sequence of the process methods, there is actually no network traffic between the victim and the server. There is a progress bar in the interface, which could generate traffic when being updated, but it is not updated during the entire computation of a chunk. Consequently, only the total processing time of a chunk through the entire stage is visible to the attacker. This seems insufficient to guess the secret output.
- the final output does not seem to have an effect on the network traffic.

Consequently, we believe there is not SCL in time in this application.

How our tools supported this conclusion:

Because this application uses the Spring framework, our tools were not able to produce an acceptable call graph for this program. Hence we proceeded by manually inspecting the decompiled source code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Another way that would allow the attacker to get the final output file would be to impersonate the victim so that it can download the final output file. However, the vaadin library generates a cookie that we think the attacker cannot discover, and that cookie is required to be able to download the output."

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.9.2 Question 025 (AC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

“We identified a vulnerability in the method process of com.bbn.hero.BeatBlock. BeatBlock is a kind of modules that can be added by the user (attacker). […] For each BeatBlock in the pipeline, this method executes a loop that iterates 44100 times. At each loop iteration, if the value of current_beat is greater or equal to 16, the loop writes to System.err. The attacker can set the BPM to a huge value such that the test is always (or almost always) greater than 16: one can take for example BPM = 400000, which makes the value of sample_period equal to 1.

Why the code contains a vulnerability:

If setting the BPM above to 400000, i.e. sample_period = 1, the execution of the method process will call the method System.err.println about 44k times, which takes a long time because the server started with ./startServer.sh writes its stderr to the console, creating expensive I/Os.

Why the vulnerability is exploitable:

A single BeatBlock is not enough to exceed the time budget of 450s, but the attacker can define a sequence of tens of BeatBlocks, each of them with a BPM = 400000, which is going to exceed the budget of 450s while still using less than 5Mb of network traffic.

How our tools supported discovery of the vulnerability:

Because this application uses the Spring framework, our tools were not able to produce an acceptable call graph for this program. Hence, we proceeded by manually inspecting the decompiled source code.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The unintended vulnerability reported by GrammaTech was identified by several other teams, and we note that it results in high CPU consumption; however, as observed by Two Six Labs, this identified vulnerability is not believed to sufficiently impact the benign request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“In the collaborative discussion, Utah mentioned they had found the same vulnerability we did. In addition, Colorado mentioned a different vulnerability in BeatBlock, relying on the fact that it is possible to trigger a large number of invocations of the fade() method which performs expensive matrix multiplications. Finally, the Iowa State team mentioned a separate vulnerability in FileInputBlock, relying on the creation of a file with a large number of channels (in excess of 32,000). However, Vanderbilt and subsequently other teams pointed out that the underlying web framework in this program is non-blocking; therefore, while it is possible to cause the server to consume excessive resources, it is not possible to prevent the server from servicing the victim user’s request. For this reason, we are changing our answer to “no” despite the presence of numerous algorithmic complexity vulnerabilities in the program.”

Live-Collaborative Evaluation:

Through collaboration, GrammaTech correctly observed that the non-blocking behavior of the application prevents a vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.10 Railyard

A.6.2.2.10.1 Question 013 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Statements on any potentially vulnerable code structure in the application:

There is a potential vulnerability in which the program might allocate an arbitrary amount of heap memory in the second call to Platform.printToStream in Platform.printCars. […]

Statements on investigations of potentially vulnerable code structure including complexities of said code structures:

In the same way as for Q018, we can achieve an infinite loop in printToStream by creating a cycle in the linked list that stores the cars of a train. This is done by adding two cars of type coal car and coal_car. See details in answer to Q018. The first call to printToStream writes the cars of a train directly into a file, so it does not require much heap memory. In contrast, the second call prints the cars of a train into a BufferedWriter that is held in memory.

Statements on why the application does not contain a complexity vulnerability (strength and reachability):

If we create a cycle in the cars of a train, the first call to prinToStream will run forever and the second call will never be reached. The only possibility of reaching the second call in this case is if fileStream is null. This variable is assigned to a file out.txt at the beginning of the programs' execution. If the creation of out.txt fails, the railyard server will be vulnerable. However, an attacker has no way to influence this.

How our tools supported this conclusion:

Our tools identified printToStream as a method that writes data to file out.txt with a high potential complexity."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"[…] However, during the live engagement one of the participants suggested that the cycle in the train could be broken by sending another train with coal cars (remember that all coal car share the same next field). Then we collaborated all together to come up with a way to exploit it.

In order to obtain the desired memory usage, we need to break the cycle in the train so the execution finishes the first call to printToStream and immediately after create another cycle so the second call to printToStream runs forever.

We can break the cycle and create a new one by sending another train in a different platform with two coal cars, one at the beginning and one at the end. When Platform.linkCars starts executing it will break the cycle and by the time it finishes there will be a new cycle. The cycle has to be broken long enough for the first printToStream call to finish but not too long so it is re-established before the second call to printToStream finishes.

For that purpose, the second train contains several extra cars. We also add cargo to the second car (with quantity 0). This slows down the execution of the second call to printToStream and gives time to the other thread (the one executing linkCars) to re-establish the cycle thereby increasing the chances of success.

The amount of data sent by the script is 163181 bytes which is under the 5 MB budget."

Live-Collaborative Evaluation:

The challenge program does not contain an intended vulnerability; however, the vulnerability path identified through collaboration of breaking the infinite looping of the intended AC vulnerability is feasible. This unintended vulnerability was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.10.2 Question 018 (AC Space, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Code that contains vulnerability:

There are four relevant methods for this vulnerability: Platform.printToStream, Platform.linkCars, Platform.printCars, and Platform.AddCar and the class Coal_Car.

Why the code contains a vulnerability:

Consider the method Platform.printToStream below. The outer loop in this method iterates over the train cars that are being sent out. It iterates over the cars using the next field of each car which form a linked list. Thus, if there is one car that points to itself, the loop will run forever. As a result, the file out.txt will grow to an arbitrary size. […]

How our tools supported discovery of the vulnerability:

Our tools identified printToStream as a method that writes data to file out.txt with a potential unbounded loop."

Take-Home Evaluation:

GrammaTech identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.10.3 Question 035 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

The secret is the materials that have been added to the most recently generated railyard schedule. In order to see those materials, a user without privileges would have to either obtain the password of a privileged user or infer the materials by observing the traffic generated by a

railyard manager. Otherwise, a user without password always gets the same response to every request.

Statements on why the application does not contain a side channel (strength and reachability):

The password checking algorithm does no seem to have significant side channels. This together with the fact that the password can be very long (e.g. passwords of 100 characters are allowed) makes it very unlikely that an unprivileged user can obtain the password with only 14 operations. The alternative would be to infer the materials from the sizes of the packets generated in a session of the railyard manager. Materials can be added to a train at any point. That means that the railyard manager can add a cargo with a new material in its last interaction of the session. If we want to obtain this information, we need to be able to extract it from the sizes of the packets of that last interaction (the information is not present before). We experimented with adding cargo with different materials and monitoring the sizes of the communication packets. These sizes did not change according to the materials. Therefore, we believe there is no side channel.

How our tools supported this conclusion:

We identified the relevant code using manual inspection of the decompiled code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.11 SimpleVote 3

A.6.2.2.11.1 Question 001 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

In order to access a ballot, a user has to sign in and provide a registration key. In contrast to previous versions, in Simplevote_3 one cannot register a key that belongs to another user. When a key is being registered, the system checks that the registration key belongs to the user that is logged in. Therefore, in order to obtain someone else's ballot, one would need to log in first (know the username and password) and then know the registration key i.e. there are two secrets: the password and the registration key. […]

Statements on any potential side channels within the application:

The time that it takes for the server to respond to a registration key depends on the value of the registration key. This is because before checking the registration key in the table, it is also checked with an algorithm whose execution time highly depends on the key. This check is in

RegistrationGate.check (in package com.cyberpointllc.stac.easydecision.manager). Therefore, it might be possible to obtain the value of the registration key by monitoring this time.

Statements on why the application does not contain a side channel (strength and reachability):

Even if we obtain the registration key, we still need the password of the target user and there seems to be no way to obtain it with the limited budget of 25 operations. There is a side channel in space for the password (see question 37) but it is not possible to exploit it with the given budget. The password checking time does not seem to provide any meaningful information.

How our tools supported this conclusion:

We identified the relevant code using manual inspection of the decompiled code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.11.2 Question 033 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

The way to obtain a voter's profile is to guess its password. The password is checked in com.cyberpointllc.stac.webcontroller.manager.TokenedPasswordChecker.processPassword which is called from com.cyberpointllc.stac.webcontroller.manager.LoginManager.handleParcel (L128). From examining the code, there does not seem to be a significant time dependency. Some experiments showed there is not a significant time difference according to the attempted password (neither in the complete query time or in the timestamp of the answer).

Statements on any potential side channels within the application:

We did not find any potential side channels in time that could allow obtaining a user's profile.

Statements on why the application does not contain a side channel (strength and reachability):

The time spent checking a password does not depend on the value of the real password or on the value of the provided password.

How our tools supported this conclusion:

We identified the relevant code using manual inspection of the decompiled code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.11.3 Question 034 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on why the application does not contain a side channel (strength and reachability):

Previous versions of simplevote had a side channel vulnerability that would allow to obtain this. The vulnerability was based on two points: the number of spaces (minus 2) in the last textual answer to one of the polls would determine the threshold (in percentage) of votes needed for a candidate to be included in the summary message that is transferred in the inter-server communication. This is determined by the methods update1 and updatel in the class ResultsMessageCreator in com.cyberpointllc.stac.easydecision.aggregation. The second part is that this same answer would be used to generate the TST structure (in the method buildMessage of ResultsMessageCreator) used to compress the summary message later. That means that if this answer contains the name of a candidate, the compressed answer would be smaller.The attack consisted on submitting answers with the name of the candidate and different amount of spaces to get packages with different thresholds. At the same time, the attack would observe the size of the inter-server communication and detect for which thresholds the sizes were smaller than usual. By varying the threshold, the attacker could perform a binary search. Unfortunately, the code that uses the threshold has changed in such a way that makes the vulnerability not exploitable: […]. There does not seem to be any other way in which a user can influence the inter-server communication to obtain information about the elections.

How our tools supported this conclusion:

We identified the relevant code using manual inspection of the decompiled code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.11.4 Question 037 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"How the observables relate to the secret:

In order to get a voter's profile, we need to obtain his or her password. The real password and the given password are compressed together in com.cyberpointllc.stac.webcontroller.manager.TokenedPasswordChecker.processPassword. The length of the resulting token is used to generate a random session string in LoginManager.fetchSessionString(). That means that the length of the returned url depends on how similar the real and the given password are.

Why the observables reveal the value of the secret (even in the worst case) with a side channel that is sufficiently strong:

processPassword in simplevote3 does not always include the complete real password. They key is in: […] In the else branch, the character of the real password at position c is not inserted in the token. This branch can be taken (but it is not necessarily taken) once j reaches 18 - this.secret.length(). This code is inside a loop that iterates over all the characters of the secret password and j is only updated there. Finally, the amount padQuantity is computed as follows: […] That means that the code is vulnerable depending on the secret password's length.

For the following lengths, the secret password is always completely included and the code is vulnerable:

- 5: padQantity is 3 and (18 - this.secret.length()) is 13
- 6: padQuantity is 2 and (18 - this.secret.length()) is 12
- 8: padQuantity is 1 and (18- this.secret.length()) is 10
- 9: padQuantity is 1 and (18- this.secret.length()) is 9

For passwords of different length, the complete secret will only be stored sometimes (depending on the result of TokenedPasswordChecker.random.nextBoolean()). This makes the vulnerability harder to exploit because the same candidate can generate urls of different lengths. The longest url will generally correspond to the case where the complete secret is compressed. That means that we can repeat a query several times until we are sure that we have the longest possible response.

- For passwords of length 7 and 10, at most one character can be skipped which gives us 2 possible sizes.
- For passwords of length 11, 2 characters can be skipped which give us 3 possible sizes.
- For passwords of length 12, 4 characters can be skipped which give us 5 possible sizes.
- Etc.

In order to be sure, we have observed the maximum size, we have to observe all the sizes for each possible candidate.

Why this side channel is triggerable by an attacker. Note, if the side channel is believed to be passively persistent, a statement to this effect is sufficient:

The attack starts by sending an empty password a fixed number of times and recording how many different sizes it gets as a response. For example, assume the password response can vary in size by 1. If we make 5 requests, the probability of getting the same size in all the responses is very low 2/232=0.06 (6%). Therefore, if in 5 request we always get the same size, we can be almost certain that the password has a vulnerable size.Once we know how many different sizes there can be, we infer the characters of the password one at a time. For each character, we send the same request until we get all the possible answers and keep the longest (that is the one that included the complete password). Then, we keep character that has the shortest of those. For vulnerable sizes (5, 6, 8 and 9) we have at most 5 + 94 * 9= 896 (we have one request per

character as the size of the answer does not change). For a password of size 7, it might still be possible to obtain it within budget. On average we need 2 requests to get both sizes for each character which makes 94* 2* 7= 1316 but I am not sure it can be done with 90% probability. For passwords of size 10 or higher, it is very unlikely that they can be obtained within budget. Even on average we will need more than 1500 requests.

How our tools supported discovery of the vulnerability:

We identified the relevant code using manual inspection of the decompiled code."

Take-Home Evaluation:

We don't believe this reported side channel is vulnerable as the size of the packets are not consistent between different guesses of the same password.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After a brief discussion, we concluded that even though there is a side channel, the budget is insufficient. The worst case have to be considered, which in this case is a password of length 15. For such a length, there is a lot of variability in the length of the answer (due to the randomization) so a much higher budget would be required."

Live-Collaborative Evaluation:

Following collaboration, GrammaTech correctly concluded that their initial findings could not fit within the budget.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.11.5 Question 038 (AC Time, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Statements on any potentially vulnerable code structure in the application:

There is a complexity vulnerability in time that can cause a 300 second delay in the code that checks the registration keys provided by a user. In particular, in the method RegistrationGate.check (in package com.cyberpointllc.stac.easydecision.manager). However, this vulnerability is not exploitable in the given budget.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures:

The expensive computation seems to be the call to this.checkCore(kStr,n) with a high value of n. Than n value is obtained from the registration key (The digits are reordered using the array decoder and n is taken from the 6 least significant digits of the transformed registration key). In order to trigger this.checkCore, we need to pass through this.isInBounds and this.checkSurface which require n to be less or equal than MAX_N=995 and for n to be divisible by 7.

Statements on why the application does not contain a complexity vulnerability (strength and reachability):

In order to provide a high enough n we have to increase the limit MAX_N up to 1001 i.e. the first number above 955 that is divisible by 7. We can increase MAX_N by doing requests with n=995,996,..., 1001. The Registration keys corresponding to these numbers are: […] This

amounts for 7 requests to increase the limit+ the login request+ the request to trigger the worst case with n=1001. In total we need 8 requests. In some experiments we measured that the size of these packets is 8196. This makes the vulnerability not exploitable.

How our tools supported this conclusion:

Our tool IST identified the method RegistrationGate.checkCore as potentially relevant for time."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In the take home answer we presented an attack that could cause a running time of more that 300 seconds but we believed it was out of budget. However, Fish Wang, a team member from Northeastern, developed a modified version of the attack script that keeps the ssl connection open throughout multiple requests. As a result, the same attack that we had can now be performed with a much lower budget. The total amount of data sent by the attack measured in the reference platform is 3173 KB which is under the 4KB limit. […]."

Live-Collaborative Evaluation:

The challenge program did not contain an intended vulnerability; however, the exploit identified in the collaborative engagement was confirmed

Post-Engagement Analysis Ruling: Correct.

A.6.2.2.12 SimpleVote 4

A.6.2.2.12.1 Question 023 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

In order to access a ballot, a user has to sign in and provide a registration key. In Simplevote_4 one cannot register a key that belongs to another user. When a key is being registered, the system checks that the registration key belongs to the user that is logged in. Therefore, in order to obtain someone else's ballot one would need to log in first (know the username and password) and then know the registration key i.e. there are two secrets: the password and the registration key. The relevant code that, given a registration key, performs the additional check for the user, is in com.cyberpointlcc.stac.straightselection.guide in the method SanctionTestGuide.handleDelivery. This method calls StraightSelectionServiceActual.confirmSanctionKey which checks that the combination of user and registration key appears in the corresponding table (/content/registrationkeys.json).

Statements on any potential side channels within the application:

The time that it takes for the server to respond to a registration key depends on the value of the registration key. This is because before checking the registration key in the table, it is also checked with an algorithm whose execution time highly depends on the key. This check is in

SanctionAuthenticator.confirm (in package com.cyberpointllc.stac.straightselection). Therefore, it might be possible to obtain the value of the registration key by monitoring this time.

Statements on why the application does not contain a side channel (strength and reachability):

Even if we obtain the registration key, we still need the password of the target user and there seems to be no way to obtain it with the limited budget of 25 operations. The password checking time does not seem to provide any meaningful information.

How our tools supported this conclusion:

We identified the relevant code using manual inspection of the decompiled code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.12.2 Question 030 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Code that contains vulnerability:

A similar vulnerability to Q38 is present here. Fortunately, this vulnerability can be exploited within the budget. The relevant code is in com.cyberpointlcc.stac.straightselection in the method SanctionAuthenticator.confirm which calls this.confirmValue. The method confirmValue and calls ChaoticOperator.chaoticOperator. This method can be very costly for certain values of its parameter n.

Why the code contains a vulnerability:

This code uses a cache that stores some precomputed values. This is implemented in the class Stash (package com.cyberpointllc.stac.math). This cache memory stores some pre-computed results up to the value 988. Whenever we query chaoticOperator with a value n, the method obtains the closest value in the stored in Stash and computes the value from n starting from there. E.g. if we call chaoticOperator(990), the method will obtain the cached values for 988 and call nextValue twice. They key of the attack is to maximize this computation. […]

How our tools supported discovery of the vulnerability:

Our tool IST identified the method SanctionAuthenticator.confirmValue as potentially relevant for time."

Take-Home Evaluation:

GrammaTech identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.12.3 Question 039 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Statements on if and how the observables relate to the secret:

The secrets are the statistics of how an election is going. This is transmitted among simplevote servers periodically.

Statements on any potential side channels within the application:

The size of the packets sent in the inter-server communication is potentially related to their content which is the secret.

Statements on why the application does not contain a side channel (strength and reachability):

In this version the number of spaces (minus 2) in the last textual answer allows to determine the threshold of votes that a candidate need to be included. The relevant code is in ResultsMessageBuilder in the package com.cyberpointlcc.stac.straightselection.aggregation. This is set by update1 and updatel and used in method shouldInclude. The other key aspect of the vulnerability is to obtain packages of different size depending on the candidate selected. Unfortunately, in this case the TST structure in the method buildMessage is build based on the summary string sumStr first and the last textual message later. This sumStr is what is later compressed with the TST structure so the resulting size will be bigger as more candidates are included in the summary. The size of these packets is also affected by the attacker's answer. Therefore, the attacker can observe that the size of the messages changes for certain thresholds and it also changes depending on the last textual answer. However, we have not been able to find a clear relation between those changes in size and the percentage of voters that each candidate has.

How our tools supported this conclusion:

We identified the relevant code using manual inspection of the decompiled code.”

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.12.4 Question 041 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Statements on if and how the observables relate to the secret:

The way to obtain a voter's profile is to guess its password. The password is checked in LoginGuide.handleDelivery in the package com.cyberpointllc.stac.netserver.guide. This method obtains the user with the given username and calls Member.matches (in com.cyberpointllc.stac.netserver) which compares both username and password. From examining the code, there does not seem to be a significant time dependency. More precisely, the execution time of the method com.cyberpointllc.stac.netserver.Member.passwordsEqual() has an execution time that only depends on the length of its two arguments. This could allow the attacker to guess the length of the password, but not its value. Some experiments did not find a significant time difference according to the attempted password (neither in the complete query time or in the timestamp of the answer).

Statements on any potential side channels within the application:

We did not find any potential side channels in time that could allow obtaining a user's profile.

Statements on why the application does not contain a side channel (strength and reachability):

The time spent checking a password does not depend on the value of the real password or on the value of the provided password.

How our tools supported this conclusion:

We identified the relevant code using manual inspection of the decompiled code.”

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.13 STAC Coin

A.6.2.2.13.1 Question 007 (AC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Statements on any potentially vulnerable code structure in the application:

The server responds to a GET */balance/me request by executing wallet.getMyBalance(). If the attacker can get wallet.getMyBalance() to take a long time then a GET */balance/me would also take a long time.

Statements on investigations of potentially vulnerable code structure including complexities of said code structures:

wallet.getMyBalance() works by iterating over all the public addresses owned by the user and summing up the balance associated with each address. The accesses and summations of the balances each take constant time. This means the amount of time it takes the server to execute wallet.getMyBalance() is linear in the number of keys owned by the victim.

Statements on why the application does not contain a complexity vulnerability (strength and reachability):

As stated the execution time of wallet.getMyBalance() depends on the number of public addresses owned by the victim. The number of public addresses owned by the victim cannot be affected by the attacker. This is because the only ways to add public addresses is through GET /wallet/:walletPassword/import and GET /wallet/:walletPassword/getNewAddress, which both require the password of the victim. This shows that an attacker cannot affect the execution time of wallet.getMyBalance(). An attacker could slow down a GET */balance/me request by occupying the thread that would service a GET */balance/me request with some other task. However, this does not seem possible. The server has two types of threads. The has one miner thread, which only tries to mine blocks and thus does not have any direct impact on wallet.getMyBalance(). The server also has threads which service requests. Unfortunately for the attacker the server spawns a new thread each time a request is made. This means that if an attacker crafted a request to busy up a thread, it would only busy up the thread servicing the attackers request. The thread servicing the victim request would not directly be affected.

How our tools supported this conclusion:

Our tool IST heuristically identified wallet.getMyBalance() as a relevant method for execution time."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.13.2 Question 008 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No*)*

Take-Home Response:

"We would like to note our interpretation of the budget for this question. For this question there is a client server and an attacker server. During normal execution of the application the attacker server will make HTTP requests to the client server for which the client server will provide a response. Similarly, the client server will make HTTP requests to the attacker for which the attacker server will provide a response. The question text for this question states that the budget is the sum of the sizes of the requests from the attacker server to the client server. Thus, we interpret the data in the responses from the attacker server to the client server to NOT count towards the budget.

Code that contains vulnerability:

The relevant methods for this attack are Miner.run, Miner.announceBlock, and Miner.tryToUpdateFromPeer.

Why the code contains a vulnerability:

If the victim miner in Miner.run mines a new block, the miner extends its local block chain with the newly mined block. If this extension increases the length of its local block chain the miner will call Miner.announceBlock to alert the network of a newly mined block. In Miner.announceBlock the victim miner will make a request to the attacker server inquiring about the length of the attackers block chain. If the attacker states that it has a longer block chain, then the victim will make a series of requests to obtain an extension so the block chain of the victim will agree with the attacker's block chain. Once the victim obtains the extension from the attacker it will extend its own block chain with the newly requested blocks. The victim will then write the newly extended block chain to the wallet.dat file.

Why the vulnerability is exploitable:

The attack works as follows. On startup, the attacker begins creating valid transaction to cycle staccoins between its own addresses. Once a valid transaction is created the attacker begins to mine a new block with the new transaction. Upon completion of mining the attacker does NOT announce the new block to the client server. Instead the attacker repeats this block creation process until it has a block chain long enough which would result in a wallet.dat file to exceed 1GB. Once the attacker has such a local block chain, the attacker makes a valid transaction request to the client server. The attacker then allows the client server to successfully mine the block with the requested transaction. This will cause the client server miner to call Miner.announceBlock, which will request the block chain length of the attacker. The attacker will respond with the length of the block chain it locally created. If this length is longer than the length of the block chain local to the client server, the client server will then request an extension from the attacker. The attacker responds with the blocks in the local block chain it constructed, which was unknown to the client server up until this point. The data the attacker sends to the client server will greatly exceed the budget of 1MB; however, all this data will be sent via attacker responses. Thus, under our interpretation of the budget, none of this counts towards the 1MB limit. Upon receiving the new extension, the client server will write the new, very large, block chain to wallet.dat in Miner.tryToUpdateFromPeer.

This attack relies on the ability of the attacker to create a local block chain which is longer than the block chain local to the client server. This can be achieved by making sure the client server does not mine any blocks while the attacker is constructing its local block chain. If during this construction the attacker does not provide any new transactions to the client server, the only way for the client server to extend its own block chain is when the client user issues a /spend request. In this case the client server will create a new transaction and attempt to mine it. However, the attacker can prevent this mining, because in StacCoin.java, before miner.receiveTx(newTx) is called HTTPUtil.sendHTTPSPost(reqURL, genericGson.toJson((Object)newTx)) is called. Within HTTPUtil.sendHTTPSPost, a post request will be sent from the client server to the attacker server. The attacker can cause the thread servicing the spend request to block on conn.getResponseCode() by just not providing a response to the new transaction request. Thus, the attacker can prevent the client server from mining any new transactions. This will ensure the attacker can locally create a sufficiently long local block chain, which is guaranteed to be longer

than the client's block chain. We expect it will take the attacker a few weeks to compute a sufficiently long block chain; however, this is all offline computation.

How our tools supported discovery of the vulnerability:

Our tool IST heuristically identified Miner.tryToUpdateFromPeer as a relevant method for file size."

Take-Home Evaluation:

GrammaTech mis-interpreted the budget calculation. All requests sent from the attacker device (attacker NUC) to the server NUC are counted as part of the budget any request not required for the attack should not be sent to avoid counting towards the budget. Effectively requests required to maintain the connection are counted as part of this. We believe that this exploit exceeds the input budget limit. The team made it clear in their response that the interpretation of the budget definition motivated the vulnerable response. Therefore, this has been changed to a "No" response and evaluated accordingly

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

*A.6.2.2.13.3 Question 042 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes, Live: *Yes)*

Take-Home Response:

"We have crafted an attack which allows an attacker to steal money from the victim. The protocol which underlies staccoin contains a vulnerability. Specifically, the signing mechanism for a transaction is not sufficient. A transaction consists of inputs, which reference unspent outputs, and outputs which state where the staccoins are to be sent. The application then signs the inputs, using a private key. This is appropriate. Essentially, when someone wants to spend staccoins they release coins assigned to their public address, by signing the inputs of a transaction. However, the issue is that the outputs of a transaction do not have any impact on the signature of a transaction. In other words, a signature releases coins associated with unspent outputs, but a signature does not sign where those coins will be release to. For example, consider the given example transaction. Here the victim creates a transaction which sends 1 staccoin to the attacker and INT.MAX-1 staccoins to a change address owned by the victim. The transaction is valid because the signature of the transaction allows INT.MAX coins to be released. However, the privileged user is not establishing where the coins will be sent. Thus, consider an attacker who receives the example transaction. This is a valid transaction. However, the attacker can arbitrarily modify the outputs of the transaction and the transaction will still be valid, because the outputs don't affect the signature. More concretely, the attacker can create a valid transaction which transfers all of the victim's coins to himself, by simply modifying the outputs of the example transaction to send INT.MAX staccoins to the attacker address. To actually steal the

coins the attacker then needs to ensure that his modified, but still valid, transaction will be mined before the original unmodified transaction.

How the observables relate to the secret:

We interpret the secret of this attack to be a valid transaction which sends the attacker more than the authorized 1 staccoin from the example transaction. The observable then is the unmodified example transaction, i.e., the valid transaction which sends 1 staccoin to the user and INT.MAX-1 staccoins to the victim's change address.

Why the observables reveal the value of the secret (even in the worst case) with a side channel that is sufficiently strong:

The attacker can construct a valid transaction which sends INT.MAX of the victim's staccoins to the attacker, by modifying the outputs of the original transaction to consist of a single output which sends INT.MAX of the victims staccoins to the attacker. Since the outputs of a transaction does not affect the signature of a transaction, the modified transaction is still valid.

Why this side channel is triggerable by an attacker. Note, if the side channel is believed to be passively persistent, a statement to this effect is sufficient:

This side channel is passively persistent. The attacker completes the attack, by mining the modified transaction. The attacker can ensure that he will mine the modified transaction before the victim mines the original transaction, by preventing the original transaction from being added to the Miner.txQueue. This is because the original transaction was created by a spend request. In StacCoin.java, before miner.receiveTx(newTx) is called HTTPUtil.sendHTTPSPost(reqURL, genericGson.toJson((Object)newTx)) is called. Within HTTPUtil.sendHTTPSPost, a post request will be sent from the client server to the attacker server. The attacker can cause the thread servicing the spend request to block on conn.getResponseCode() by just not providing a response to the new transaction request. In this case the original transaction will never be added to the Miner.txQueue. Thus, the attacker is guaranteed to win the race to mine the new block. Once the attacker mines the modified transaction, the network then agrees that the victim sent INT.MAX staccoins to the attacker.

How our tools supported discovery of the vulnerability:

We discovered the vulnerability through manual inspection of the decompiled code.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. GrammaTech identified an unintended out-of-scope vulnerability to spend another user’s STACCoin as requested by the challenge question.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“For the take home portion we provided an attack which allows an attacker to steal money from a victim. After talking with the Northeastern and Utah team, we believe our attack is out of scope. This is because our attack is extracting information from the main channel rather than a side-channel. However, our attack still works and we would like to present it as an unintended vulnerability.”

Live-Collaborative Evaluation:

While GrammaTech's response was "No", they identified an unintended out-of-scope vulnerability therefore this response was changed to a "Yes" as the team identified a vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.14 Swappigans

A.6.2.2.14.1 Question 002 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Code that contains vulnerability:

The code containing the vulnerability is in the class com.weathers.swappigans.transaction.SwappigansItemMatcher. This class contains the logic to compute the subsets of the user items that have a price close to the purchased item. […]

Why the code contains a vulnerability:

This code recursively builds all possible subsets of the user's items, but tries to simplify/discard subsets when the price is too close to another subset. Consequently, The execution time of the algorithm increases significantly with the number of user items, but also varies a lot depending on the item's prices. In addition, the subsets are constructed recursively by always adding the item that was added first by the user, which means that the order in which the user adds items for sale is important.

Why the vulnerability is exploitable:

The idea is to create two users: the first user sells a very expensive item (i.e. the maximal possible price), and the second user adds a lot of different items with prices s.t. simplifications in the algorithm are not possible. The following code is a subset of the python script for the attack, which is a modified version of the provided python script swappigans_demo.py […] When running this script, the total outgoing network traffic is between 34 and 35kb according to the field length of the tcpdump output, and the time to run the last request findBestTradeForItem is greater than the budget of 6s.

How our tools supported discovery of the vulnerability:

Using the settings in IST to identify ACVs in time, the subproblem 3 reported by our tool was consisting of the two methods mergeTrimRemGreater and apxSumRecursion. From there, we reached our final answer by manual inspection of these two methods."

Take-Home Evaluation:

The challenge contains an intended vulnerability that enables the attacker to exceed the item price. GrammaTech identified an unintended vulnerability that was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"There seems to be an attack that uses the similar idea as our attack from the take-home, but it requires less budget. In the take-home, it seems like we needed 34 small-priced items. Other teams needed less than that (e.g. only 29) with almost similar settings (credits to the Colorado team):

expensive item is $1000.00

small items are created as follows:

price of each item is half of the price of the preceding item

minimum price of 1 cent

items are added in decreasing price order”

Live-Collaborative Evaluation:

GrammaTech correctly found the intended vulnerability and their exploit was strengthened after talking to other teams.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.14.2 Question 020 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“How the observables relate to the secret

In this question, we assume that the attacker knows the username of a victim. In order to be able to make a purchase with the victim's items, it is sufficient for the attacker to know either the victim's password (in this case, the attacker can just login with the victim's username and password to obtain its session token), but it is also sufficient to know only the session token. If the attacker has the session token of the victim, it can make a purchase request with the username and session token of the victim.

Why the observables reveal the value of the secret (even in the worst case) with a side channel that is sufficiently strong, and why it is triggerable by an attacker:

The problem is in the class SessionTokenManager, that contains the logic to create and check the session tokens of the users. […] The session token is generated using an AES cipher initialized with SessionTokenManager.key, and only depends on the userName, which is known to the attacker. Consequently, it is sufficient for the attacker to know the value of SessionTokenManager.key to obtain the victim's token. This key is created with new SecretKeySpec(SessionTokenManager.secret.getBytes(), "AES"), and SessionTokenManager.secret is the return value of getSaltString(16). If the attacker knows the return value of getSaltString(16), it can then simply reproduce the computations to generate the victim's token. getSaltString is using an unsecure random number generator, Random.nextFloat(), seeded by System.currentTimeMillis() / 1000L. With the knowledge of its own username and token, an attacker can try all possible values for (System.currentTimeMillis()-k) / 1000L, k>0, incrementing k until the computation of the token returns its own token. This computation can be done offline and finds the right value of k in less than 15min if the server has started less than 10 years ago. One can then use this k to generate correct session tokens for any username, and impersonate the victim.

How our tools supported discovery of the vulnerability:

The vulnerability was discovered by manual inspection of the decompiled code.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The potential unintended vulnerability reported by GrammaTech will be investigated further is believed to represent an unintended out of scope vulnerability. In future challenges we will clarify that the attacker does not know when the server was started.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“In discussions, it emerged that the vulnerability that we have discovered is a degenerate case of an SCL in time and space, in the sense that it does not even require any information from time or space observations. In other words, the vulnerability is present, but, strictly speaking, whether or not we can qualify the attack as an SCL attack is not entirely clear.”

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.14.3 Question 031 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“We understand the observable as being the execution time of the method:

com.weathers.swappigans.http.requests.ItemPurchase.getResponse(). Indeed, when the victim user wants to purchase an item, it sends an ItemPurchase request to the server, and the server computes the set of the victim's items used for that purchase. This takes some time depending on the prices of both the purchased item and the victim's items. Once the computation is finished, the server creates the HTML page with the relevant information and sends it to the victim. Because the computation time of this method depends on the price of the purchased item, i.e. the secret, it is likely that there is some leakage of information.

Statements on any potential side channels within the application:

As mentioned above, an attacker could guess for which item a victim has made a purchase request by analyzing the time between that request and the server's reply. Everything in the getResponse method seem to have a relatively constant time except for the call to com.weathers.swappigans.transaction.SwappigansItemMatcher.calc(Integer, Double). With the help of the attack from question 020, the attacker can even know the list of items for sale by the victim. This gives him the exact set of prices relevant to the execution time of the calc method.

From there, the attacker could tentatively create items with the same prices in his account, and try to send purchase requests to the server while monitoring the response time. Then, it could guess which item was purchased by the victim by comparing the execution time of the requests.

Statements on why the application does not contain a side channel (strength and reachability):

We believe that it is not possible for the attacker to craft an attack that would work with 85% probability of success and only a 32 operations budget. The above approach would likely need much more than 32 operations to try all possible purchase requests. Moreover, even with a higher budget, it is unclear whether the execution time of the requests would be enough to disambiguate all the possible purchased items in the database, as many of them will have relatively similar prices.

How our tools supported this conclusion:

We reached this conclusion by manual analysis of the decompiled code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.15 Tollbooth

A.6.2.2.15.1 Question 011 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"We have identified two reasons why this application could be vulnerable. In the following, we detail these two reasons:

Reason 1: Each peer in the network has its own packet queue. The class com.bbn.PacketQueue defines the UUID of the root node, as well as the UUID of the peer. When it receives a packet, the peer checks whether the destination UUID is equal to its own UUID, and if not it forwards the packet to the appropriate destination: […] The method com.bbn.PacketQueue.sendPacket(TollBoothPacket) that is used to forward the packet makes use of the routing table stored as a field in the PacketQueue object. However, if the destination UUID of the packet is not in the routing table, the packet is broadcasted to everyone in the network. We believe there could be a way to generate a very high network traffic by sending packets with UUIDs that do not belong to any peer in the network. These packets would be broadcasted indefinitely by all the peers, which would create a DOS attack on the root node and prevent other requests to be processed in time.

Reason 2: We have identified a possible race condition in the class com.bbn.TollBoothRootWorker. This class has a field private Map<Integer, VehicleRecord> balances that tracks the balance of the registered transponders. The field is allocated as a HashMap, which is not thread-safe and require proper use of synchronize keywords to make sure it is not read and written at the same time by different threads. All methods in the class com.bbn.TollBoothRootWorker are synchronized except the method handlePacket.

This method reads the balances map when it handles a RequestTransponderList packet:

```
this.packetQueue.sendPacket(new
TransponderListPacket(tbPacket.getSource(), this.balances.keySet()));
```

Because the handlePacket method is not synchronized, it can potentially execute at the same time as other methods in this class, e.g. addTransponder, that adds new keys and values to the HashMap. With some probability, the concurrent execution of HashMap.put() and HashMap.keySet() can lead to incorrect execution of the TollBoothRootworker and potentially

an ACV-T. The attacker can send many requests to create new transponders using the web interface (note that these requests are not pushed to the packetQueue, so they can execute at the same time as a request pulled from the queue). At the same time, it can send TransponderListPacket requests to the packetQueue of the root worker. This is possible if the attacker controls a leaf peer. However, we expect that, for this attack to work in practice, the attacker would need to send a very large amount of requests. We have measured experimentally that a request to create a new transponder using the web interface requires more than 1kB of outgoing network traffic. Because the budget in this question is only 5kB, we are unsure that this attack will have a success rate of 50%.

How our tools supported discovery of the vulnerability:

We obtained this answer after manual inspection of the decompiled code."

Take-Home Evaluation:

The challenge contains an intended vulnerability. GrammaTech reported a potential unintended vulnerability. We believe that the input budget is insufficient for Reasons 1 or 2 to be viable exploits.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"During the live engagement we further explored our first intuition from the take-home engagement - the fact that it was possible to flood the system with broadcast packets with an unknown destination. However, we found that a packet with a destination that is not in the routing table is not resent.

On the other hand, if we send the root node a packet with null destination, this raises a NullPointerException in the worker thread […]

This exception is not handled, so it crashes the thread. All subsequent benign requests from leaf nodes will be ignored; this places the system in a state that meets the formal specification of the ACV question. Consequently, our answer is still that a vulnerability is present. It is worth mentioning that we can also send a packet with a null source and crash a different thread […]

Live-Collaborative Evaluation:

GrammaTech identified an unintended vulnerability to prevent the root node from servicing the benign request to be delayed.

Post-Engagement Analysis Ruling: Correct

A.6.2.2.15.2 Question 024 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Statements on if and how the observables relate to the secret:

In order to transfer funds from a target transponder, we need to obtain either the password or the token for such transponder. The token is computed with a hash function over the user id and the password. The observables in this question are the communication between a leaf node and a root node when a charge_transponder request is executed.

Statements on any potential side channels within the application:

The communication of between a leaf node and a root node when charging a transponder only involves the transponder id. It does not involve either the password nor the token. The relevant code is ToolBoothRootWorker.handlePacket and TollBoothRootWorker.chargeTransponder in the root node, and TollBoothWorker.carPassed and TollBoothWorker.handlePacket in the leaf node.

Statements on why the application does not contain a side channel (strength and reachability):

Because none of the methods involved in the communication refers to either the password nor the token, it is not possible that the sizes of the packets or their timing depends on them. Therefore, we conclude that there is no vulnerability.

How our tools supported this conclusion:

We identified the relevant code using manual inspection of the decompiled code."

Take-Home Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

GrammaTech correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3 Iowa State University

A.6.3.1 Iowa State Overview

Iowa State answered 38 questions with the highest R&D Team accuracy of 76% during the take-home engagement – second highest overall behind the control team. Iowa, GrammaTech, Colorado, and Vanderbilt all achieved between 73% and 76% accuracy in the take-home. Against the segmentation of answered questions, Iowa state was in the leading group the R&D Teams, and was approximately even in performance with GrammaTech, with both teams within 3 percentage points of one another in accuracy, TPR, and TNR.

Iowa achieve a 100% accuracy in side channel questions during the take-home but struggled with AC Time questions with only a 30% accuracy. The predominant reason for these incorrect AC Time questions was due to incorrectly assessing the complexity bounds of identified complexities. Against the segmentation of all questions, Iowa State tied with Vanderbilt for the second highest R&D Team accuracy of 67%. The team had the 4 highest TPR (50%) and TNR (78%) in the segmentation of all questions.

As with Engagement 5 Iowa achieved a 100% accuracy in the segmentation of answered questions following the live-collaborative engagement. The R&D Teams achieved accuracies between 97% and 100%. Against the segmentation of all questions, the team's 88% accuracy ranked fourth, marginally trailing Vanderbilt. Collectively, the R&D Teams identified 3 unintended vulnerabilities during the live collaborative engagement – Railyard Question 013, Class Scheduler Question 017, and SimpleVote Question 038. Railyard Question 013 and Class

Scheduler Question 017 came out of a collaborative effort from all teams. The SimpleVote Question 038 unintended vulnerability was confirmed with a clever exploit by Northeastern.

A.6.3.1.1 Take-Home Engagement Scores

Table A-125: Engagement 6 ISU Take Home Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	7	7	100
SC Time	9	9	100
SC Space/ Time	4	4	100
AC in Space	8	6	75
AC in Time	10	3	30
Total	**38**	**29**	**76**

Table A-126: Engagement 6 ISU Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	15	8	53
Not Vulnerable	23	21	91
Yes Answer	11	8	73
No Answer	27	21	78

A.6.3.1.2 Live-Collaborative Engagement Scores

Table A-127: Engagement 6 ISU Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	7	7	100
SC Time	9	9	100
SC Space/ Time	4	4	100
AC in Space	8	8	100
AC in Time	10	10	100
Total	**38**	**38**	**100**

Table A-128: Engagement 6 ISU Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	15	15	100
Not Vulnerable	23	23	100
Yes Answer	16	16	100
No Answer	22	22	100

A.6.3.2 Iowa State Specific Responses

A.6.3.2.1 BattleBoats 3

A.6.3.2.1.1 Question 022 (AC Space, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"From the graph it is clear, that there are no recursive calls within the interesting part of the app.There are only 4 loops contained in the 4 methods. We examined them individually. The loop in HitPointer.parse is parsing the parameters of the shoot cannon operation. It does not allocate a new structure. Hence, it is ruled out. The loop in updateHitsAsSunk creates a new HashMap to update the Hits. As the map is not static it would be garbage collected at the end of this operation. Using dynamic analysis, we confirmed that a maximum of four squares can be hit in one shot. Each entry in this HashMap can be of maximum size of 23 bytes. So, this loop will not allocate more than 92 bytes. Hence, it is ruled out. In sqrt method, there is a loop that creates multiple instances of BigDecimal variables; however, these variables are local and will get garbage collected upon return of the method. Hence, the loop is ruled out. Which leaves us with the loop in determineHitTime. Shown below is the code as shown by ourdecompiler view: […]

The above loop grows the size of the trajectory a HashMap consisting of BigDecimals by adding new entries. Each newly added entry in the HashMap (i.e., Key-Value pair) has a total size of 32 + 32 = 64 bytes. Therefore, increasing the memory consumption. Based on this analysis we selected the loop in determineHitTime as the likely candidate for AC Space vulnerability.

Hypothesis: The loop in HitPointer.determineHitTime can be triggered by an attacker and can lead to consume memory exceeding the usage limit.

Strength and reachability of the vulnerability: Using taint analysis, we confirmed that the attacker's input reaches the potentially vulnerable loop. In this case, the inputs being the parameters of a shot - height, velocity, elevation and, board angle. Next, we used our dynamic analysis capabilities to generate inputs which could exceed the memory consumption beyond the usage limit. To maximize the resource consumption, we used to set the board size to maximum (20 from the description. We verified that it is not possible to set the board size greater than 20). In particular, we tried three approaches:

1. Different combinations of the four parameters:
   Each of the four parameters have limits placed on them. We initially tried all the integral value combinations, as they form a finite input space. We did not find a quadruple which exceeded the resource usage limit.
2. Floating point precision:

The input values can be fractions. We hypothesized that for a highly precise floating-point number the loop iterates several times and makes the HashMap grow beyond usage limit. We observed that it does increase the energy consumption. The more precision we add, more we exhaust the input budget (as the input request string grows in size). From our analysis, we could not find an input which is within the input budget (10kB) and exhausts the usage limit.

3. Try sending parallel requests:
   Another way to tackle this problem would be to run multiple instances of this operation. In order to do it, we need to send a shoot cannon request before the previous request is processed. If the previous request is processed, then the turn passes to the benign user and all other requests would be discarded. We were not able to successfully achieve this as the loop termination is very fast. So, we could not get it to run multiple instances simultaneously.

Based on the above analysis we concluded that this app does not contain an AC Space vulnerability with the given description."

Take-Home Evaluation:

Iowa correctly flagged the intended vulnerable method in computing hit time, along with the vulnerable behavior of the HashMap growth; however, the team's inability to discover an instance to trigger the vulnerability led to an incorrect response.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"At the collaboration we learnt of an attack that works within the specified budget. Hence, we are changing our answer to "Yes". During the collaboration, team Colorado and team GrammaTech shared nearly identical exploits. While the exploits worked, they could not provide a satisfactory justification as to why they worked. Team Colorado shared their exploit scripts with us which we used to further analyze the application. These scripts are included with the report. The exploit sets up a game and shoots with the cannon (three shots to be precise). This is a fuzzer generated exploit. The fuzzer has come up with three "magic" numbers which make the exploit work. They are

1)Size of the square

2)Height

3)Velocity

We tried varying the numbers and check if the exploit still worked. It consumes 1.1GB after the first shot itself. We observed that it does not matter if we modify the height or the size of the square. It is only the velocity value which drives the exploit. Our second observation was that the memory consumption depends upon how many digits to the right of the decimal point the velocity has. More digits, more memory consumed. We performed taint analysis for the velocity […]

We observe that the HitPinpointer.terminate calls HitPinpointer.sqrt which depends upon the velocity. It also depends upon the height but the only effect it has is if the height is greater than zero it does the expensive computation based on the velocity. Hence, the height should be strictly greater than zero. The important piece is HitPinpointer.sqrt accepts a BigDecimal with possibly

10,000 digits, converts it to a double value, passes it to Math.sqrt and converts it back to a BigDecimal. We believe that this operation ends up in a large amount of memory consumption which is the cause behind this exploit. Based on this analysis we are changing the answer to YES."

Live-Collaborative Evaluation:

Following collaboration, Iowa was able to correctly answer that the program was vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.1.2 Question 026 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"If the attacker can discover the location of their cannon in fewer than 36 operations, they could exploit it to discover all boat positions within budget. […] There was one place where a very weak side channel in space was found to leak part of the cannon location. Square.pullEncoding() encodes the coordinates in spaces. This is appended to the "end placing boats" message. However, before doing so it passes through a length filter twice. This renders the filtered message either a 2 or 3, most cases being a 3, rendering the side channel too weak to exploit."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.1.3 Question 040 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"If the attacker can discover the location of their cannon in fewer than 36 operations, they could exploit it to discover all boat positions within budget. […] There was one place where a very weak side channel in time was found to leak the border of the ocean during shot calculation, but it is likely too weak to exploit in practice. During shot calculation, OceanScreen.pullOceanSquares() has a sanity check to determine if the shot is within the ocean board. If the shot is out of bounds, it returns early, potentially skipping expensive computation. Observing the timing difference in the response from the opponent, an attacker could infer the cannon location using a binary search on each axis. In practice, empirical testing shows that the remaining expensive calculations in pullHits() and updateHitsAsSunk() are likely too fast to use as a side channel in practice (0.2 to 0.6ms)."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Thanks to our collaborators on question 40, who pointed out that the short circuit we noted was only distinguishing between on or off radar. Regardless of timing, this branch is of no use in determining the canon position. We have re-written the code for clarity and included it below. This does not change our answer, but it does simplify it. During the live engagement, we started a separate investigation based on Northeastern's observations. Northeastern had noted a potential timing difference between on / off ocean screen based on measuring packet timings in wireshark. To help answer this question, we instrumented the server binary manually to record timings around 1) the distance calculation, and 2) the hit calculation. While meeting with all teams, Northeastern et al performed manual experimentation with the instrumented binary, on the Nucs. As a group we decided that there was too much noise to exploit the observed differences, which still excluded network effects, even assuming there was a pattern to exploit."

Live-Collaborative Evaluation:

Through collaboration Iowa confirmed their non-vulnerable response.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.1.4 Question 043 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Placing the cannon takes place on the benign user side. There are two ways to place the cannon: 1) provide specific location coordinates, and 2) randomly select the location. The operation itself takes place in the method Battleboats.defineCannon. […]

The entire call chain tries to ensure the location of the cannon is not occupied by any other boat. There is no way to control the random placement of boats and the loops have a fixed number of iterations. Hence, it is not possible to get a timing difference within this operation. […]

Shooting a cannon is an operation under the control of the attacker. The attacker provides four parameters to make a shot: Height, Velocity, Elevation, and Board angle. We analyzed for the methods involved in processing the response for this particular operation.

There are no recursive methods and only four methods containing loops: HitPinPointer.parse, HitPinPointer.determineHitTime, HitPinPointer.sqrt and OceanScreen.updateHitsAsSunk. HitPinPointer.parse is a routine parser that goes over the list of parameters provided. The loop has a fixed number of iterations. Hence, we ruled it out.

OceanScreen.updateHitsAsSunk goes over the squares that were hit and updates their status as Hit in a loop. From the description, the only way to hit more than one square is to get to land on the edge of a square. At best, it can land on the intersection of four squares. Which means this loop will never iterate more than 4 times. While it depends upon the parameters controlled by the attacker, it does not relate to the observable. Hence, we concluded it can't cause the vulnerability on its own. HitPinPointer.determineHitTime contains a loop which can be controlled by the attacker (confirmed using Taint analysis) and also contains an interprocedurally nested loop (loop in the method HitPinPointer.sqrt). What is more interesting is, its result is used to determine whether the shot landed within the board or not. This is checked in the method OceanScreen.pullOceanSquares. […] pixelX and pixelY depend upon the result of the suspicious loop and the coordinates of the cannon. If the branch condition returns true, it does not process the hit and return immediately.

We can predictably get the cannon to be shot outside the board and due to the existence of the branch condition specified above a timing differential is created, leading to discovery of the secret. Using dynamic analysis, we discovered that when the parameters height and elevation are set to 4 and 3 respectively, the distance of the shot is roughly equal to the velocity. Which means, we can predictably determine the xDistance and yDistance. If we set the board angle to 0 degrees or 180 degrees then the distance of the shot = xDistance. If we set the board angle to 90 degrees or 270 degrees then the distance of the shot = yDistance. According to our hypothesis, we should observe a consistent timing difference when the shot lands outside the board. Since, the origin of the shot is from the cannon, it means we observe the timing difference when, xDistance= x coordinate of the cannon (with board angle 0 or 180) yDistance= y coordinate of the cannon (with board angle 90 or 270) In order for this line of attack to work, we had to confirm the hypothesis -shooting outside the board creates a predictable timing difference. Based on our experiments, we were unable to observe such a timing difference."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.2 BraidIt 3

A.6.3.2.2.1 Question 004 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The vulnerability is triggered by sending a message that the new modified braid is "ȺȾʂʐɐɑɒɥɦɱɻȺȾȾ"and that all 5 of the initial braids are "yyy". This is not possible to do if we use the challenge program as the client but it is possible by using the Python file "interact.py" provided with the problem as a malicious client.

The steps to trigger this are as follows:

1. Offer game with 10 strands (this is expected by the benign script)
2. Wait for the server to send modified braids
3. Respond to the server with the outcome
    a. The outcome may be either true or false
4. Send the malicious braids
    a. The modified braid is "ȺȾʂʐɐɑɒɥɦɱɻȺȾȾ"and each of the original 5 braids is "yyy"

When the victim receives the modified braids, it will attempt to check if "ȺȾʂʐɐɑɒɥɦɱɻȺȾȾ"is equivalent to "yyy". This causes the infinite loop which logs the message "After free reduce: YYYȺȾʂʐɐɑɒɥɦɱɻȺ". This app uses the Log4J framework to write log files. The Log4J

configuration of the challenge program causes 4 log files to be created, each of size 10 MB. This results in 40 MB of files, exceeding the target of 25 MB."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. The unintended vulnerability reported by the team was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.2.2 Question 027 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Our tool was able to identify the secret (plaitNum) in SelectPlaitCommand class and the Plait object created which stores the selected plait. […] Our tool was able to identify the place where packets were sent back by player B to player A after player B selected the secret. Since this is a peer to peer app, Our tooling was able to identify methods in the io.netty.Channel package as possible application entry points or communication interfaces. We were able to identify the Dispatcher and the SelectPlaitCommand class, the execute method of which is executed, when a plait (secret) is selected. We were also able to identify the DeliverModifiedPlaitCommmand class which sends the modified braid and the original braids back.

The DeliverModifiedPlaitCommmand class uses a builder to build the modified String and all the 5 braids. The executeEntity method constructs this. The first part is the selected braid which is obtained from the PlaitSelected.takePlaitString method. Our backward Taint Analysis revealed that the length of this String might depend on the selected plaitNumber. […]

The taint on PlaitNum in SelectPlaitCommand class revealed a differential branch point where in one branch(branch1), the plaitIndex was set to the secret (the selected plait Number), and in another branch(branch2), that was always set to 1. Our analysis shows that in fact the plaitIndex is always set to the selected plaitNumber.

The method take PlaitString()in the PlaitSelectedPhase class returns the String corresponding to the selected Plait. The length of the String returned is always the length of the selected braid. Although the taint shows that the secret (plaitIndex) might influence this

length, it turns out that this is not the case; the length of String is only multiplied by (plaitIndex / plaitIndex) which is 1, so the length returned is not influenced by the secret (plaitIndex). The plaitNum selected by the opponent is then used to retrieve the Plait in class SelectionPhase. Even in this scenario, our tool did not find any potential side channel where the length of the selected braid could be influenced by the PlaitNum. Due to this, we conclude that there are no potential side channels concerning the PlaitNum."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.3 BraidIt 4

A.6.3.2.3.1 Question 032 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Our analysis did find a side channel in space. When playerB sends back the modified braid to playerA, the whole packet consists of the modified braid plus the 5 braids. The length of the modified braid part of the packet depends upon the selected braid. Specifically, it is (modified braid length times the braid number). The attacker knows the lengths of all the braids, and the length of the modified braid, so from the packet size the attacker can figure out the index of the braid selected by the user. However, when the braids are presented to the attacker they are shuffled, so the index of the braid may no longer match the braid selected. Because of this, we conclude that this side channel cannot be exploited.

Our tool was able to identify the place where packets were sent back by player B to player A after player B selected the secret. Since this is a peer to peer app, Our tooling was to identify methods in the io.netty.Channel package as possible application entry points or communication interfaces. We were able to identify the Dispatcher and WeaveItDispatcher as a dispatcher class and from there the SelectWeavecommand class, the execute method of which is executed, when a weave (secret) is selected. We were also able to identify the TransferChangedWeaveCommmand class which send the modified braid and the original braids back. […]

Following the taint backwards and using our program dependency analysis smart view, we concluded that the weaveNumber is only set in certain case, and that is dependent on result of SelectionState.takeErrorState(), and that it is only set when SelectionState.takeErrorState() != null. Continuing our analysis, we can see that this can only occur if errorStatevariable is not null. Backward taint analysis reveals that this is always the case. The errorStatevariable is initialized to empty string (not null) and if an error occurs it is set to an error String.

[…] Does this side channel reveal the secret? No. However, the weaves are randomly shuffled. Because of this the attacker cannot figure out which braid was selected, just based on the "n" above."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“Our previous analysis showed that all other team justifications for yes answers were incorrect. It was argued by other teams that the weave shuffling occurs on the client side and a malicious client could be developed that simply does honor the shuffling operation. This however was contested by ISU that the shuffling occured on the victim side and was not controlled by the attacker (as described in this report). To disprove the malicious client hypothesis, ISU created the malicious client and performed the attack (shown below). The malicious client (included with this report) is indicated in red and simply removes the shuffle statement from SelectionState.prepareForSelections method. As can be seen from the screenshot, the malicious client still receives the weaves in a shuffled order which invalidated the hypothesis of every other team.

After this conclusion the prediction of the random number generation routines was explored further during group collaboration. The Weave.grabRandomWeave method is responsible for randomly generating the weave strings a random number generator. The same instance of the random number generator is later used to shuffle the 5 weaves. A previously noted side channel exists, which allows the attacker to learn the index of the originally changed weave, however after shuffling there is no correlation with the chosen index. If it is possible to recover the unshuffled ordering of the weaves then it is possible to exploit the application.A collaborative analysis of the Weave.grabRandomWeave (between all teams) resulted in an observation that some information about the state of the random number generator is leaked through the values of the strings computed for the 5 weaves.

Initially this was thought to be not enough information to recover the random seed, but GrammaTech recovered a project that claimed to support prediction on Java nextInt(range) method (instead of simply nextInt() or nextLong() each of which leak significantly more information and are handled by tools such as ReplicatedRandom). With the JavaCG projectsome promising experiments by GrammaTech showed that it may be possible predict the characters after the strings were generated for a given weave. GrammaTech refined this proof of concept to account for the calls to nextBoolean (which updates the state of the random number generator, but only results in whether or not a character is uppercase or lowercase and is not needed otherwise for prediction value). Using this ISU helped to devise an attack scenario where the original ordering can be recovered. By predicting the next string that could be computed a relative ordering for each weave can be recovered with high accuracy (short strings do not provide enough information to accurately predictthe next sequence). By considering the long strings that map to sequences that do not exist the last weave can be identified. Afterwards each relative ordering can be established with respect to the last weave and the original unshuffled order can be recovered enabling the original attack outlined in this report.A proof of concept code to predict the next random weave strings (developed by GrammaTech), and the modification to JavaCG to enable accounting for the call to nextBoolean are included with this report as evidence for the exploit.”

Live-Collaborative Evaluation:

We believe that GrammaTech’s exploit to undo the shuffle procedure is feasible within the operational budget. However, as the java random library is out-of-scope, we have noted this as an out-of-scope vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.4 Calculator 1

A.6.3.2.4.1 Question 014 (AC Space, Unintended Vulnerable, Take-Home: Yes Live: Yes)

Take-Home Response:

"The vulnerable code in the app is in the method com.cyberpointllc.stac.cruncher.processEquation().

How we identified the vulnerable code using our tools:

We audited the call graph of the application for each of the mathematical operations to find hotspots.

1. Call graph for multiplication operation was particularly interesting for 2 reasons:
    a. It was the hub for exponentiation, finding root and divide operations
    b. There is a guard that switches between two different algorithms for multiplication depending on the number of digits in the input numbers. One of the algorithms used is in Transform.transform(), which eventually calls Transform.primaryTransform() which is a recursive method calling itself over 30 times.
2. Call graph for Cruncher.processEquation() is also interesting because it is used by all types of calculations (screenshot S2), and at as a last step, this method formats the result of the calculation as a String. This led to the hypothesis that an attacker could cause the server to use a large amount of memory, if he manages to request for a mathematical calculation whose result has a large number of digits.

We instrumented the App to monitor the memory consumption for each method and ran our dynamic analysis. We discovered a drastic increase in memory consumption after calling the method Cruncher.processEquation(), especially when the calculation result is longer than 10k digits. We further analyzed the most memory-consuming operations during the runtime of JVM using queries with random numbers. We found that character sequences (Strings) uses more than 85% of the memory (screenshot S3). This matches with our above observation about Cruncher.processEquation(). It turns out that the return value of processEquation() is a formatted String with the final result of the calculation. The abnormal high memory usage could be the super long String that is generated here.

To build our exploit within the budget, we ran experiments with short queries involving exponentiation and multiplication that would evaluate to a result with a large number of digits. Specifically, the exploit described earlier sends 62 requests where each request is 964 bytes long (as captured by tcpdump), causing the server to generate a 50000 digit number. […] This application contains an AC Space vulnerability as per this question and as demonstrated by our attached exploit."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. Iowa's unintended vulnerability was confirmed by reducing the number variable from their exploit from 62 to 52 such that it satisfied the input budget limit.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.4.2 Question 016 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The vulnerable code in the app is in the method com.cyberpointllc.stac.cruncher.GreatNumber.transformMultiply.

We audited loops and recursive calls in the application to find hotspots.

1. Loop catalog did not reveal any suspicious loops for AC Time vulnerability.
2. Dashboard informed us of 18 methods involved in recursion, of which 11 were not reachable from the entry point for the app. We audited the remaining 7 methods
    a. Of the 7 methods, Transform.primaryTransform() and Transform.speedyTransform() were the most interesting because they called themselves multiple times. These methods were invoked during several operations including multiplication.
3. The call graph for multiplication operation was particularly interesting for 2 reasons:
    a. It was the hub for exponentiation, finding root and divide operations (screenshot S2).
    b. There is a guard that switches between two different algorithms for multiplication depending on the number of digits in the input numbers. One of the algorithms used Transform.transform(), which eventually called Transform.primaryTransform() which is a recursive method calling itself over 30 times.
4. Dynamic analysis experiments with instrumented code confirmed that GreatNumber.transformMultiply was the hotspot during multiplication.

Inter-procedural projected control graph (PCG, see screenshot S5) for the callsites to GreatNumber.transformMultiply revealed that the condition for executing this method requires that the number of digits of both multiplied numbers to be greater than 1001. Further, Transform.primaryTransform() recursively splits numbers with more than 1000 digits into numbers less than or equal to 32 digits and passes them to Transform.speedyTransform(), which contains time consuming nested loops (due to randomness in termination, as described in the previous section). However, the App restricts numbers sent in client requests to be no more than 10000 digits long. We performed dynamic analysis with multiplication and exponentiation of the form A^n1x ... x A^ nm resulting in numbers with large number of digits. We found that in general the server takes more time if n1... nm are in descending order, and lesser time otherwise. We built the exploit described above based on our analysis."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. Iowa's unintended vulnerability was confirmed by reducing the number variable from their exploit from 62 to 52 such that it satisfied the input budget limit.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.5 Calculator 2

A.6.3.2.5.1 Question 003 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"We hypothesized that vulnerabilities could be caused by (1) explicit write operations to files, (2) explicit logging by the app, (3) implicit logging when an exceptions are thrown.

We audited the relevant code corresponding to (1), (2), and (3) mentioned above using RULER's Subsystem Interaction filter. The app had 69 methods calling 92 APIs in the IO_SUBSYSTEM in total. Of the 92 IO_SUBSYSTEM APIs, only9 APIs corresponded to "write" APIs relevant to the question. This reduced the number of interactions to be audited from 184 to 25. Further only 3 of the interactions were relevant because the rest were not reachable from the entry point of the app. These3 interactions related to network write operations when the response was being sent back to the client, and they were not vulnerable.

[…] The loggers were triggered when an InterruptedException or ExecutionException is thrown. Since there was no way for the attacker to send input that would trigger InterruptedException or ExecutionException, we concluded that interactions with the Log subsystem cannot lead to a vulnerability. We also hypothesized logging APIs that may be present in the other libraries. The app had 34 methods calling 56 APIs in the OTHER_LIBRARIES_SUBSYSTEM. None of these calls related to file IO. Hence, we concluded that interactions with the Other libraries subsystem cannot lead to a vulnerability.

We hypothesized that printing of stack traces can lead to large files being written when there is an exception. We identified 4 locations in the app where stack traces from exceptions were printed out. […] In our extensive dynamic analysis for other questions related to this app, we could never trigger a write to the log file "stac.log". This supports our conclusion from static analysis that the attacker cannot provide input to trigger the app to write to a file."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"We stick to our "No" answer.

During the take home engagement we had correctly identified the code segments that causes the logging operation that writes log information to file "stac.log". We pointed out two exception handling possibilities, namely InterruptedExceptionand ExecutionException,which may write the log information to the log file. For InterruptedException, we stay with our conclusion that this exception cannot be triggered by the user with any form of user input. However, for

ExecutionException, we were able to find a way to trigger it with a stack overflow exception. During our collaboration, GrammaTech demonstrated an exploit that with an input of equation that has highly nested parentheses (9000 left parentheses followed by 9000 right parentheses), they are able to trigger a stack overflow exception to the server program which results a writing operation to the target log file "stac.log". The cause of this vulnerability is the recursive call within method Mather.calculateOutcome()(see screenshot S2 below). For every left parentheses, this method pops the first character from input equation and calls itself with the remainder of equation. This behavior would cause stack overflow when the number of parentheses gets abnormally big, which triggers the logger to write log information to the file. What's interesting is, for each time this exception is triggered, the new log is appended at the end of the file which means that we may able to reach 4 Mb file size by conducting multiple attacks one after another. With our observation, the content written to the file is 76 kb per attack, it takes at least 54 attacks to get a log file with 4 Mb in size. In this exploit, the PDU size of the application requests sent from the attacker to the server is about 20 kb per attack. Unfortunately, the maximum sum of the PDU sizes is limited to 60 kB which means we can only perform 3 attacks at most. Obviously, it is definitely not enough. With further analysis to related code segments, we believe that there is no cheaper way that a user can perform such an attack and trigger an ExecutionException. Therefore, our answer to this question is still "No".To summarize, we had enough analysis during the take home engagement to help us find suspicious code segments, but we were unable to build an exploit. The collaboration helped us refine our understanding of how to exploit the vulnerable code. However, we believe that the attack is still not able to be done within budget."

Live-Collaborative Evaluation:

The collaborative engagement helped Iowa to explore GrammaTech's potential vulnerability. The team correctly concluded that the challenge program was not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.5.2 Question 005 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The vulnerable code in the app is in the method com.cyberpointllc.stac.mather.BigInteger.plainLogspaceMultiply.

We audited loops and recursive calls in the application to find suspicious program artifacts.

1. Loop catalog did not reveal any suspicious loops for AC Time vulnerability.
2. Dashboard informed us of 19 methods involved in recursion, of which 11 were not reachable from the entry point for the app. We audited the remaining 8 methods
    a. Of the 8 methods, Reformulate.chiefReformulate() was the most interesting because it called itself 5 times.
3. The call graph for multiplication was particularly interesting for 2 reasons:
    a. It was the hub for exponentiation, finding root and divide operations
    b. There is a branch point that switches between two different algorithms for multiplication depending on the number of digits in the input numbers. One of the algorithms used Reformulate.reformulate(), which eventually called Reformulate.chiefReformulate() which is a recursive method.
4. We then decided to run experiments to multiply an n-digit number m-times in a single request. Since the server may only accept requests that are within the length of 50,000

bytes, with the format of "AAAA x AAAA x AAAA x ..... x AAAA", the relationship between n and m would be $(n + 1) * m - 1 <= 50{,}000$. Dynamic analysis experiments with values of n from 500 to 1002, and m from 99 to 49 revealed that BigInteger.plainLogspaceMultiply was the hotspot during multiplication.

We audited the control flow for BigInteger.plainLogspaceMultiply and found that there were 2 nested loops whose termination conditions depended on a coin flip (generated random number between 0 to 1 being less than 0.5). Further, dynamic analysis targeted on BigInteger.plainLogspaceMultiplyrevealed that for high number of iterations of the outer loop in this method, the randomness in termination of the inner loops does contribute significant amount of time."

Take-Home Evaluation:

The challenge program contains an intended vulnerability in parsing input expressions. The unintended vulnerability reported by Iowa was confirmed.

Post-Take-Home Analysis Ruling: Correct.

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.5.3 Question 009 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The vulnerable code in the app is in the method:
com.cyberpointllc.stac.mather.Mather.calculateEquation()

We audited the call graph of the application for each of the mathematical operations to find hotspots.

1. Call graph for multiplication was particularly interesting for 2 reasons:
   It was the hub for exponentiation, finding root and divide operations
   a. There is a branch point that switches between two different algorithms for multiplication depending on the number of digits in the input numbers. One of the algorithms used Reformulate.reformulate(), which eventually calls Reformulate.chiefReformulate() which is a recursive method calling itself 5 times.
2. Call graph for Mather.calculateEquation() is also interesting because it is used by all types of calculations (screenshot S2), and as a last step, this method formats the result of the calculation as a String. This led to the hypothesis that an attacker could cause the server to use a large amount of memory, if he manages to request for a mathematical calculation whose result has a large number of digits.

We instrumented the App to monitor the memory consumption for each method and ran our dynamic analysis. This revealed a drastic increase in the memory consumption after the call to the method Mather.calculateEquation(), especially when the calculation outcome is a number

that is longer than 10k digits. We further analyzed the most memory-consuming operations during the runtime of JVM. Our analysis shows that the character sequences (Strings) uses more than 75% of the available memory (screenshot S3). This matches our observation about Mather.calculateEquation().

It turns out that the return value of Mather.calculateEquation() is a formatted string with the final result of the calculation. The abnormal high memory usage could be the super long string that is generated here. We ran our dynamic analysis on the App to multiply multiple two large numbers. The analysis shows that the number of digits of the calculation result equals to the sum of the number of digits of the two multiplied input numbers. For example, if we multiply the numbers: A and B, where A has 25000 digits and B has 24000 digits, the calculation result will have (25000+24000=49000) digits. Given that the server accepts a maximum query length less than 50000 digits long, our dynamic analysis led us to specially crafted combinations of numbers, which increased the memory usage up to 4.2 GB."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The unintended vulnerability reported by Iowa was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.6 Chessmaster

A.6.3.2.6.1 Question 006 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"We did not find a side channel strong enough to leak the secret. The attacker can measure the time taken for each move and the time taken to generate a response for the move by the server. Using this observable, the attacker can determine each move and can trace the moves to decipher the current state of the board.

[…] Our analysis found that determining the next move is performed using an artificial intelligence algorithm, which definitely can take varying amount of time depending on the move. Thus, we explored the call chain for determining the next move. The method GameClient.negamax is suspicious. It contains a loop and is recursive. Depending upon how many times it iterates, it is going to take a variable amount of time. […]

Using the information we gathered, we tried to formulate an attack. It takes one operation (the passive operation) to analyze the timings for packets corresponding to the moves which lead to capturing of pieces. Based on this information we confirmed that we could guess which piece was captured. This information though is not sufficient to solve this question. The timing differential observed is not dependent upon the positions of the pieces, hence we cannot reliably detect the positions. Since, there are 13^64 positions possible for a chess game, it is not possible

to just exhaust oracle queries to detect the positions within the specified probability of success. Hence we conclude the absence of the vulnerability with the operational budget."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"We are sticking to our "No" answer. During the take home engagement, we correctly identified the suspicious code in GameClient.nextMove which in turn calls GameClient.negamax which contains the potentially vulnerable code. Based on our analysis, we concluded that the victim moves are pre-determined, in other words the AI computing the next move is deterministic. During the collaboration Vanderbilt brought out the possibility of identifying the sequence of moves that was played in a game. Given the number of moves, the number of move sequences in a Chess game can be very large, but finite. They argued that based on the time taken by the AI to play a sequence of moves, the sequences can be divided into groups. By measuring the time taken for the entire game duration, the attacker can guess which group the game (sequence representing the game) belongs to and then by using the oracle queries the state of the game can be found out by playing the sequence. We find three problems in this line of attack:(1) For a game with small number of moves played (up to 30), this strategy may work as the number of sequences is manageably finite and as argued by Vanderbilt, the groups of sequences can be enough small for the oracle queries to be used. But not enough evidence was provided to instill the confidence that this can work for games with large number of moves (>1000). (2) As clarified by the White Team at the start of the collaborative discussion, the maximum number of moves allowed for this game is 10000. Team Vanderbilt conceded that for such a large number of moves this attack is unlikely to work. (3) There was no discussion of probability of success of this attack. Due to above three points, we don't think this line of attack will work within budget. To summarize, during the collaboration we learned of a different way of exploiting the vulnerable code we had identified. Based upon the discussion and the clarification about the number of moves from White Team, we conclude this line of attack cannot work within the budget."

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.6.2 Question 015 (AC Time, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"We discovered a potential AC Time vulnerability in the App but we could not exceed the resource usage limit within the given budget. […] Our analysis of the control flow graph revealed an interesting point: unless all the moves are processed, client does not receive any response from the server.

Interestingly, on all paths the first move has to be processed before the response is sent back to the client. This sounds unusual. Further, we found that the all the moves must go through the loops in Autocorrecter.correct and Autocorrecter.edits to be validated. Curiously, the server uses

the same instance of the Autocorrecter object for all the games. Which should stall any request for some time.

[…] This game has no mechanism to block the repeating moves nor does it place any restriction on the number of moves (we confirmed this using Taint analysis). More importantly, the history tree which AI using to predict moves is growing with every iteration of this process, lengthening the time it takes to process the requests. Thus, it would cause some delay before the new game request is processed. We can trigger this attack, hence it is reachable. The question remains about the strength. We maximized the number of moves we could send within the budget. Waited until the last move begins to be processed and then sent the benign request. We could not get the delay to exceed the usage limit on the reference platform. Hence, we concluded this is not a strong enough vulnerability."

Take-Home Evaluation:

Iowa incorrectly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After the collaboration we're flipping our answer to Yes. As detected by our tools Gameclient.nextmove and Autocorrecter.correct make up the vulnerable code. GrammaTech and Colorado brought a way to exploit this code to our attention. High level attack strategy is as follows.

1)The score of king is disproportionately high (20000) which can be learnt in the method State.totalPieceScore. 2)Note that there is a way to promote a pawn to king, enabling two kings of the same color in a game (which is not allowed in regular chess). In Server.serve, there is a check that prevents a pawn from promoting to king if the request is of the sort [move a7 a8 king]. But if an incorrect move is sent (with a spelling error), the Autocorrecter.correct tries to correct it but after that it is not checked if the pawn promotion is creating a new king or not. Hence an incorrect request is sent "[move a7 a8 kin]" which autocorrects to "kin" to "king" -The list to Autocorrecterkeywords is specified in the class Server.3) GameClient.negamax, the suspicious code we identified depends upon the variable "depth" which in turn depends upon the result of State.totalPieceScore. If the attacker (who always plays as white) creates a new king, then the total score shoots up and depth increases making GameClient.negamax take a long time. 4)If a benign user tries to play a game after that, the server does not respond to the newGame request in time. Team Colorado had an automatic exploit which they shared with us. We tested their exploit on the reference platform and confirmed the vulnerability."

Live-Collaborative Evaluation:

Following collaboration, Iowa identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.7 Class Scheduler

A.6.3.2.7.1 Question 012 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"There are two possible side channels that should be audited in this App.

Hypothesis 1: The login mechanism does not sufficiently prevent active operations against authenticated user resources (e.g., download generated schedule).

Hypothesis 2: Passive observations of network traffic between a benign authenticated user and the server form a time based side channel that reveals the number of students and teachers in the uploaded data file.

Hypothesis 1 is quickly refuted by auditing the usage of the Authentication.isUserLoggedIn method, which returns true if the user is authenticated. The active operations that relate to the secret are all guarded by a condition that prevents requests from unauthenticated users from completing.

Hypothesis 2 involves passively observing network interactions between a benign user and the server, which may include uploading a data file, generating a schedule, or downloading a generated schedule. Examining the uploaded data file reveals that the data file is an XML file that contains entries for teachers, students, and courses along with their attributes. Since the size of the XML file would grow with each entry and the secret is the sum of teachers and students it may be possible to estimate the secret by the size of the uploaded file. However, there is a problem with this attack: 1) it relies on knowing the network bandwidth (if bandwidth is known, the file size could be computed by the time it took to upload the file), and 2) it assumes that the start and end of the file upload could be identified solely through timing observations. Our analysis did not indicate any of these conditions are true. […]

There does appear to be a side channel in time that reveals the time it takes to complete a schedule generation task and the number of students and teachers can influence that time, but this side channel is not strong enough to reveal the secret within the attack budget."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Utah noted that unauthenticated requests can be made to the progress bar status endpoint. Our previous analysis did note that the progress bar leaks the total time to complete the course schedule generation, to which we found no strong correlationand concluded that the side channel was not strong enough. After considering Utah's analysis, we reconsidered our previous side channel by considering active operations to query against the progress bar update status. An active operation allows an attacker more insight into the generation process by observing the progress bar was at percentage X at time 1 and percentage Y at time 2 therefore

giving the attacker information about the rate the progress bar was advancing (vs. previously the attacker only knewthe total time).We performed a dynamic analysis to confirm that the endpoint was accessible from an unauthenticated user running a different browser. We discovered that 1 active operation is required to recover the assigned brower JSESSIONID, CSRF token, vaadin instance ID. It is also necessary to correctly specify the synch ID (which starts at 0 for new sessions). Successful Unauthenticated Request to Progress Bar StateWith this information we performed a targeted dynamic analysis to record an ideal amount of information (using more than the budget allows) about the progress bar status over time during various combinations of students and teacher school data files. We recorded the network traffic and parse the progress bar

state along with the timestamps. A collaborative observation that the full precision of the progress bar state (ranging from 0 to 1 as a decimal) could reveal the numerator and denominator pair of the calculation was made. A static analysis revealed that the numerator corresponded to the generation while the denominator corresponded to the max generation (which ranges from 500 to 1500). ISU modeled and experimentally determined that over 1000 random selections of generations and max generations in the allowed range we could correctly predict the generation and max generation 70% of the time. However, since we are observing the progress bar over time and know that generations only remain the same or increment we were able to increase the accuracy of our successive predictions to 89.8% accuracy. With this knowledge an attacker could with high accuracy compute the time it took the generation algorithm to progress from generation to generation over time.

The next step was to show that no correlation existed between the number of students and teachers and the progress or speed that the generation algorithm progressed. Using a few samples of student and teacher combinations we found the algorithm to progress linearly from generation to generation over time. This ruled out that the information was being leaked over time due to a change in rate of the algorithm. However, the most damning information for the theory was that repeating the generation for the same data set (ex: 122 students, 12 teachers, 500 max generations) both produced linear lines with slopes that varied. Due to the varying slopes (whose ranges varied enough to be indistinguishable from other combinations of student and teacher numbers) for the same datasets we ruled out the hypothesis that the rate of the progress bar could correspond to the number of students and teachers."

Live-Collaborative Evaluation:

Iowa correctly concluded that the response time contains too many collisions to resolve the secret within the input budget.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.7.2 Question 017 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"It is possible to upload a valid school data file that crashes the generate schedule thread (with nearly 100% reliability) preventing the request for downloading the generated schedule from ever completing. The size of the malicious school data file is approximately 1kB, which well within the attack budget of 500kB.

[…] By interacting with the app, we learnt that 1) the button to upload new school data is labeled "Upload School Data", 2) the button to generate a new schedule is labeled "Generate Schedule", and 3) once a schedule has been generated a button labeled "Download Generated Schedule" is created. A brief analysis provided in class_scheduler_sampler.sh bash script also reveals this information and also indicates that in the example scenario the benign request can only complete after the "Download Generated Schedule" button appears.

A taint analysis on the conditions values that govern each relevant branch reveals that in order for the Download Generated Schedule button to result in the download of a file: 1) the user must be logged in, and 2) a previous schedule must have been successfully generated. If condition 2 is not satisfied, it is still possible to download a previously generated schedule although a message is displayed that indicates the schedule cannot be downloaded (this does not block the return result of the previously generated schedule). This behavior could be used to write a no answer

for this question by one interpretation of the question because the question asks about the "Request to download the most recently generated school schedule (demonstrated in class_scheduler_sampler.sh -line 234)", however inspecting the class_scheduler_sampler.sh makes it clear that the question is referring to schedule that is generated for the uploaded school data file (not the previously successful generation of a school data file) since it keys off the event of the first appearance of the Download Generate Schedule button. From this information, it is clear that if the task performed by the "Generate Schedule" button exceeds 50 minutes (the benign request will exceed resource usage threshold).

[…] Some manual effort to search for possible uncaught exceptions revealed that it is possible to cause a NullPointerException in the CourseClass constructor. The null pointer is caused by a combination of the RandomNumber.getRandomNumberhelper method, which returns 0 if parameter n >=parametern2, and that the CourseClass constructor expects a value range between 1 and n, but the malicious input file results in a value of 0 outside of the expected range. This could occur for example when a data file contains a single teacher, which results in a random selection of 1 to 1 random teachers, which returns 0 causing the teacher object to be null.

We built the exploit described above based on our analysis."

Take-Home Evaluation:

We do not believe that the exploit reported by Iowa delays the response to the benign request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Colorado noted that VaadinUI.generationInProcess variable controls whether or not a new generation can begin, however this variable is an instance variable so if it was possible to create a new instance of the VaadinUI the generation could be triggered again in another thread. Colorado's exploit, like ours, was invalidated by the question clarification made onsite. Our previous exploit crashed the generation thread, but by creating negatively weighted course sections a guard can be bypassed for the maximum sum of sections, courses, teachers, etc. In this way it is possible to create a long running thread process, which is non-blocking. Our dynamic analysis was able to confirm that by opening a new browser tab (which creates a new set of sync ids, login session, and most importantly a new VaadinUI instance), the generation can be triggered again. Using Colorado's initial exploit (which has been included with this report), ISU was able to confirm the vulnerability dynamically by performing the following steps. 1) In Browser A (Chrome), login to the application 2) Upload the exploit.xml school data file 3) Press the generate schedule button 4) Repeat steps 1 and 3 (omitting step 2), four more times

5) In a separate browser (curl or Firefox) navigate to the homepage and note that the server is unresponsive ISU Utah provided the analysis of the VaadinUI instance creation to verify the exploit worked in the provided attack scenario."

Live-Collaborative Evaluation:

The challenge does not contain an intended vulnerability. This unintended vulnerability to crash the server was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.7.3 Question 028 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Our analysis revealed three possible side channels that need to be audited.

Hypothesis 1: The course schedule generation algorithm reveals a teacher's schedule through an unauthenticated time side channel.

Hypothesis 2: A timing differential exists when an unauthenticated user directly requests a teacher's schedule, student schedule, or course room assignment (all of which are dependent on each other).

Hypothesis 3: The login mechanism does not sufficiently protect the teacher's schedule or can be bypassed using login queries within the attack budget.

[…] Passive attacks are out of scope for this question, so refuting this hypothesis involves showing that the schedule generation algorithm cannot be initiated, or the progress of the algorithm cannot be monitored from an unauthenticated account. Using RULER we search for callsites to the isUserLoggedIn method. We learn that the VaadinUI$1.buttonClickis the button handler for the generate schedule button which invokes the isUserLoggedIn method (which we expect to see because this is an authenticated user task). […] The control flow graph of the button click handler is quite large, so a compact Projected Control Graph (PCG) is requested containing only the relevant events (the initiation of a schedule generation and the check if the user is logged in). From the PCG, we learn that the schedule is only generated if the user is logged in. Finally, a dynamic analysis of the application during the schedule generation revealed that status updates about the generation progress came at regular 250 ms intervals. Statically we searched for the value 250 and found that the VaadinUI.startProcessor method sets the update interval to a fixed 250 ms interval, with a call to setPollInterval(int)>(250).

[…] This hypothesis [2] assumes that an unauthenticated request can be made to interact with the application. However, inspecting the VaadinUI class reveals that the displayCalendar method, which accesses the generated schedule details for teachers, also invokes the isUserLoggedInauthentication check. It is clear that the calendar cannot be accessed without the user being logged in, so hypothesis 2 is invalidated

[…] The login functionality must be triggered in response to the request made to the login page, which contains a button with the name "Login". In RULER, we search for the logic containing the string "Login". […] A forward call graph from the LoginPage methods the reveals the LoginPage.buttonClick handler which invokes the Authentication.authentication method responsible for checking the username and password. This code reveals that there is a single hardcoded user, which is assumed that the attacker does not know. There are two differentiating branches, but neither of the branches introduce any significant time differential that could be observed by an attacker. With an attack budget of 30,000 operations and no constraints that reduce the search space of a valid username and password it would be infeasible to attempt brute forcing the username and password. Since we cannot bypass the login mechanism with the given attack budget and the active operations and we are not able to interact as an unauthenticated user with the secret (hypothesis 2), this hypothesis is also invalidated."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.8 Effects Hero

A.6.3.2.8.1 Question 010 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“we hypothesized that the vulnerability can be triggered by: (1) observing a substantial difference in timing/induced response of network traffic when downloading the final processed file, and (2) the construction of the output file enforces a specific format and restrictions on the stored data that allows the attack to reconstruct the processed file. Based on our analysis of the App, we concluded that the App did not contain the SC vulnerability in time described in the question.

[…] Hypothesis 1

The attacker can issue multiple requests to download a file by guessing the download path for the processed file returned to the target user within the given operational budget. Given that no other user is interacting with the system at the time of the attack, the correct download path will cause a substantial difference in timing/induced response of the network traffic. Thus, granting the attacker access to download the processed file returned to the target user.

Our dynamic analysis of the App using the example scripts provided lead to the knowledge that the Download Button Id can be narrowed down to be within a small range of Ids based on user Id. Moreover, the ResourceId is always equal to (Download Button Id + 1)

With the help of our dynamic analysis, we crafted multiple experiments:

1.The target user and the attacker use the same Browser Session […]

2. The target user and the attacker use different browser sessions […]

Our analysis found that each request is authenticated via a session sent as an HTTP header with the outgoing request. […] this App does not contain the SC vulnerability in time as per this question except for the case where the target user and the attacker use the same browser session which we render as out of the scope of vulnerabilities for this App.

Hypothesis 2

The construction of the output file employs specific format and restrictions on storing the data that enables the attacker to reveal the stored data bytes and eventually reconstructing the processed file returned to the target user by observing some timing difference in network traffic.

Using the Dashboard, we found that FileOutputBlock class is creating a new instance of ByteArrayInputStream with the final name "Output.wav" in generateSoundFile method. Further analysis, we discovered that the StreamResource object fed to the FileDownload is created upon

constructing a new instance of FileOutputBlock in method generateSoundFile. However, the actual construction of ByteArrayInputStream corresponding to the processed file will only occur when the user clicks the download button. This is because the first parameter passed to constructing a new StreamResource is a lambda function that is not evaluated except upon downloading the processed file. Using the Type and Callsite catalog, we have been able to identify the methods performing write operations to the ByteArrayInputStream which includes the methods: process and generateSoundFile. So we started investigating both methods to learn the internals of the output file and see if the attacker can reverse engineer the processed file by guessing data bytes within the operational budget. […] With all the randomness that can be generated through the uploaded file and the processing blocks and the wide range of numbers that can be written, it is hard for the attacker to reverse engineer and construct the processed output file returned to the target user even with observed timing difference in the network traffic.

We refute the hypothesis 2 and conclude that this App does not contain the SC vulnerability in time as per this question."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: <Correct/Incorrect>

A.6.3.2.8.2 Question 025 (AC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"We discovered that it is possible for an attacker to trigger the AC vulnerability in time specified in the question within the specified budget, exceeding the resource usage limit, and a 100% probability of success. The attacker can trigger the vulnerability by uploading a specially crafted WAV file multiple times that will cause the server to perform intensive processing that results in blocking/delaying benign users to initiate their session.

[…] Hypothesis: The attacker can upload a specially crafted WAV file that triggers the AC vulnerability in time and cause the server to perform intensive processing that results in blocking/delaying other benign users to initiate their session.

To locate the code segments pertaining to this hypothesis, we used the Dashboard and Loop Catalog to find all the loops the contain operations that correspond to I/O APIs and are user-input controlled. We found that there are only 16 loops of such kind. Thus, we suspect that the attacker can influence these loops. Using Atlas Shell, we constructed a subgraph where the root nodes corresponds to the possible entry points of the App (which the attacker can externally invoke) and the leaves are the methods containing the 16 loops with I/O APIs.

The subgraph reveals that the attacker can control any of the loops contained within the process method of the classes: BeatBlock, FileOutputBlock, and FileInputBlock. The FileOutputBlock

class contains the logic to write and download the final processed sound file. The loop in method FileOutputBlock.process() is not interesting as it has a fixed number of iterations of 44100.The BeatBlock class contains the logic to perform signal processing on the uploaded WAV file. BeatBlock.process method has a loop that calls BeatBlock.fade method that contains two loops. Using Loop Catalog and Taint analysis, we found the the loops in BeatBlock.fade method have fixed number of iterations: one that is limited to 6 iterations and the other one is limited to only 105 iterations. We also found that the loop in BeatBlock.process method have fixed number of iterations equals to 44100. Therefore, we ruled out the possibility that BeatBlock.process call chain contains the vulnerability. Finally, we are left with the class FileInputBlock which contains the logic to read and process the WAV file uploaded by the user. Using Loop Catalog, we found that the FileInputBlock.process method calls other methods in FileInputBlock class that contains loops; forming couple of deep nested loops. Most of these nested loops are user-input controlled dependent on the uploaded WAV file by the user. Hence, we suspect that this particular part of the code contains the vulnerability as per our hypothesis.

[…] Using Loop Catalog to investigate the loops contained in the call chain that originates from the FileInputBlock.process method, we found that there are 12 loops in this call chain. Five of these loops have fixed number of iterations equals to 44100. The remaining 7 loops are in FileInputBlock.getNextFloat method that is called within a loop in FileInputBlock.getRaw method. The number of iterations for each of the 7 loops is bounded by the number of channels of the uploaded WAV file by the user. […] Given that the loop in FileInputBlock.getRaw method calls the FileInputBlock.getNextFloat method 44100 times and FileInputBlock.getNextFloat calls the ByteBuffer.get()method by the number of channels, then attacker can upload a specially crafted WAV file with a large number of channels to trigger the AC vulnerability in time as per our hypothesis.

Experimenting with crafting WAV files, we were able to create a 30 seconds long WAV file using a sample rate of 44100 with 32767 channels at a sample width of 1 Byte. […] the attacker can add the File Input Block 50 times and upload the crafted WAV file 50 times which amounts to 1638 KB of data which is way below the input budget of 5MB; leaving the attacker with ~3000 KB for other button clicks and header requests. With 50 uploads of the specially crafted WAV file, the ByteBuffer.get method in FileInputBlock.getNextFloat() will be called: (4.335 * 10^7) * 50 = 2.168∗10!"times. This would mean that each ByteBuffer.get() method call would have to take less than !"#!.!"#∗!!!"=2.08∗10!!"or 0.2 Nanoseconds to pass the 450 seconds (the resource usage limit) which is impossible in real setup. Hence, we confirm our hypothesis and conclude that the attacker can trigger the AC vulnerability in time described in the question within the input budget, exceeding the resource usage limit, and with 100% of probability of success."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. Iowa identified an unintended out-of-scope (due to external library allowing invalid input) vulnerability; however, this is not believed to sufficiently impact the benign request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We are changing our answer to No. From information gained from the team GrammaTech, we believe that the Library VaadinUI protects from other sessions being started due to its internal

security. This causes other users to initiate a session, but the session does not block from other's interaction. We also believe that to exhaust the session threads would require 200 sessions. None of the vulnerabilities we found could create 200 separate sessions within the input budget. From the collaboration, other exploits include a manipulation of the BeatBlock class matrix multiplication / calculation of the matrix determinant. An exploit team Utah found was taking advantage of the BeatBlock's input source by linking each added BeatBlock to the BeatBlock after it and modifying the BeatBlock properties to have a 330,000 BPM with 1 ms Beat Rate with several sparse beats in a beat matrix. This would cause the BeatBlocks to activate twice every time it was run and add to the exploit's time. The second team, Colorado, had a separate attack of creating BeatBlocks with certain patterns which would cause a time vulnerability similar to the affected source that Utah found."

Live-Collaborative Evaluation:

Following collaboration, Iowa correctly concluded that the challenge program does not contain a vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.9 Railyard

A.6.3.2.9.1 Question 013 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Hypothesis 1: There exists loop(s) which consume enough memory to exceed the usage limit.

We discovered that the entry point of the App, Main.main uses lambda expressions to dispatch requests to various handlers. In order to find all the loops reachable from the entry point, we selected the handle method corresponding to the operations that the attacker can perform. Namely, attach and detach the train cars, view and modify of the train schedules, update the inventories (the cargo material) on each train car, and set the personnel on each train.

Methods in this graph contain 20 loops. We are interested only in the loops which contain the callsites that trigger increase in memory consumption (e.g., instantiation, adding to a data structure such as Map). We used Loop Catalog to select the callsites contained within these 20 loops then used the Type and CallSite catalog to select callsites that either instantiate or add to a data structure. We retained only the loops containing the selected callsites. There were 7 such loops […] the attacker input can control only three loops. These loops are in the methods Schedule.patch, Platform.linkCars and Platform.buildMaterials. […] Based on out analysis of these loops, we hypothesized that there is a possibility to inflate these HashMaps beyond the usage limit within the given budget.

Theoretically, we can send 542 requests to the App which can add up to 50MB of memory consumption per [request]. Therefore, we theoretically expected this scenario to easily exceed the memory usage limit. But, we discovered that the garbage collector efficiently cleans up any unused memory. In fact, we were not able to exceed the 200 MB of total memory usage. Thus, we concluded that it is not possible to do so. Hence, we refuted Hypothesis 1.

Hypothesis 2: There exists a recursive method with unguarded recursion depth that inflates memory consumption beyond resource usage limit.

This App has only one recursive method: Util.checkDirectory […] contains a loop. But, this loop is not user-input controlled and the only way the attacker can control the recursion depth would be via a loop or a branch condition which controls the recursion depth. Using the Loop Call Graph, we discovered that this method is called within a loop in the method Util.getClassesForPackage. But, this loop is not user-input controlled by the attacker. Thus, the attacker cannot control the recursion depth. Hence, we refuted Hypothesis 2.

We found evidence that invalidates both of our hypotheses. Hence, this App does not contain an AC Space memory vulnerability as described in this question."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"During the take-home engagement all Blue Teams except Team Utah had answered No. During the collaboration, the "Yes" answer from Team Utah was discussed and everyone came to an agreement that their attack cannot be done within budget. Continuing on, we and team Colorado brought a potentially vulnerable code to everyone's attention. This code was identified by us as a vulnerable code for Q18. It is a loop in Platform.printToStream method.

In Q18 we built an exploit based on the fact that this loop can run infinitely, and it writes to a file endlessly. This makes use of the fact that the field "next" in CoalCar is a static field.Team Colorado brought to our attention that this infinite loop can be broken by creating a new schedule. Once you create a new schedule, the Platform.printToStream is called again but with a different argument. The argument determines which stream it writes to (the highlighted code). This method is called from the method Platform.printCars.

When the method printToStream is first called, it operates on a fileStream object which we exploited for Question 7. After returning, printCars again calls printToStream but this time with a stringStream object. Stringstream does not write to a file but populates a stream with objects. This will work towards the memory. The high-level line of attack is as given below, (1) Schedule the train on platform A which creates a cycle like in Q18. (Taking advantage of the fact "next" field in CoalCar is static) (2) Schedule the train on platform B which also creates loop as discussed in step 1. This in turn breaks the cycle created by the schedule on platform A by resetting the static field "next" of CoalCar and Platform.printToStream method is again called by the Platform.printCars. (3) If the second printToStream call returns before the cycle is created on platform B, then the attack is rendered meaningless as the loop will not run infinitely. (4) Hence, we want to stall the second printToStream call until the creation ofcycle is complete on platform B. We believe this can be done. The details of the strategy are discussed in step 5(5) In order to stall the loop in printToStream, the following loop within the do-while loop (Loop 1) in printToStream is critical. This loop runs till all the cargo materials are processed.

(6) Hence, we create a schedule on platform B with large number of cargo materials, which stalls the second printToStream call long enough so that the cycle can be created on platform B. (7) After that the do-while loop (Loop 1) in Platform.printToStream (again runs infinitely and keeps populating the stream with new objects. We built the exploit to verify the attack. It is included with the report. Hence, we are changing our answer from NO to YES"

Live-Collaborative Evaluation:

The unintended vulnerability determined from the live collaboration was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.9.2 Question 018 (AC Space, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"It is possible to craft an input by adding cars of a specific type such that the input causes an infinite loop where each iteration of the loop logs to file

This particular app has 20 loops. Using the loop catalog, we narrowed it down to 14 loops that interact with a file. Furthermore, as the non-monotonic loops have more complex termination conditions we further prioritized the audit to non-monotonic loops. There are two such loops identified by the loop catalog. We manually examined these two loops. Of the two loops, one loop in the method Util.checkJarFilereads from a file and does not write to a file, which could not cause the vulnerability we are looking for. The remaining loop, in the method Platform.printToStream, writes to a file and consists of three nested loops, which increase its complexity and made the loop a strong candidate for containing the vulnerability.

There are three places where currentObject is updated.

1) Initialization, where currentObject is initialized with the object of the first car in the railyard.

2) currentObject is updated with the field value of the previous car in the railyard.

3) currentObject is assigned null value when an exception is thrown.

Based on this, we figured out there are only two ways to terminate the loop.

1) The loop reaches the end of the list corresponding to the cars in the railyard.

2) Loop encounters an exception.

[…] As we want to keep the loop running, so that it continuously writes to a file, we do not want to throw an exception and we want to create a never-ending list of the cars. The way to do it would be to create a list with a cycle in it. We were able to create such a list and verify the vulnerability.

[…] By systematically auditing, the locations where the currentObject field is updated, we discovered that there 8 types of cars, each having the next field. Among these 8 types of cars, only the class Coal_Car has a static next field. This discrepancy means that all the instances of Coal_Car are going to point to the same car. We used this observation to build a list with two Coal_Car instances with the type names "Coal_Car" and "Coal Car" and a cycle between them."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.9.3 Question 035 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Our analysis did not find a side channel in space with enough strength to recover the secret within the given operational budget. […] we found 2 POST requests that must be synthesized to confirm the possibility of the attacker revealing the secret: Add Cargo Material and Finalize Schedule.

[…] we created an interprocedural PCG by selecting the nodes the belong to the Taint of the returned results from Cargo.AddCargo method as events. […] The result of the Cargo.AddCargo method will be embedded within the response to the request as a JSON String […] Given the returned response to the request has a fixed size in both cases (true or false), the attacker cannot observe a substantial difference in the sizes of the responses to the Add Cargo requests, therefore, the attacker cannot reveal the secret.

We also checked whether the size of the Add Cargo Material request can help the attacker to reveal the added cargo material. The payload of the request consists of: the material name, quantity and description. While the user should choose from a predefined set of material names, multiple material names have the same length. Moreover, the quantity and the description fields can vary in size. Hence, the attacker cannot reveal the secret by observing the sizes of the Add Cargo Material requests.

[…] the user can issue a POST request to finalize the train schedule. It turns out that the request to finalize the schedule is of fixed length. Therefore, the attacker cannot use the request size of this request to reveal the secret.

Next, we investigate if the induced response size from this request allows the attacker to reveal the secret. For the attacker to reveal the secret, the response size of this request should be deterministically related to the added cargo material. However, we know from the provided example scripts and App description that the response to the finalize schedule request includes other information about added cars to the train. Therefore, the attacker cannot use the response size of the Finalize Schedule request to reveal the secret."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.10 SimpleVote 3

A.6.3.2.10.1 Question 001 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"There is a potential side channel in the code which processes the registration code, RegistrationGate.check() which could leak the registration code in time. […] Registration keys are 12 characters; when unscrambled the last 6 characters form a number n, which (to be a valid registration key) must be divisible by 7, and the first 6 characters have to match a function of n. The expensive processing of the code is repeated n times, which is how the registration key is leaked in time (determined by observation of the server response time).

EasyDecisionServiceImplementation.checkRegistrationCode() checks that the entered registration key is associated with the voter. Viewing the ballot requires entry of the registration key, and the registration key must match the voter who is currently logged in, which implies that the target voter's password must also be obtained. Per Questions 33 and 37, the password cannot be obtained in space or time. If the attack were otherwise enabled, then the side channel strength would depend on how distinct the processing time is for each value of n.

[…] An audit of the user-controlled input using taint, especially password and registration code, were used. In addition, custom queries were written to search for and audit uses of BigDecimal, which lead to an audit of RegistrationGate"

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.10.2 Question 033 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"User-controlled input can reach two places of interest: LoginManager has two methods which process the password: handleGet and handleParcel.

1. In handleGet, the code ExchangeController is always null, rendering any potential side channel in the 'then' branch unreachable.
2. In handleParcel, any potential side channel in DESHelper.pullEncryptedString() is unreachable because passwordCode is always null.

The password comparison occurs in TokenedPasswordChecker.processPassword(). The password comparison processes every character, and the cost of comparison does not depend on whether a particular character in the guess is correct; this precludes a segmented side channel in time. In addition, processPassword() includes characters from the secret password with 50%

probability, further complicating an attack within budget. Worst-case secret passwords have 15 characters from an alphabet of 94 characters, which implies a minimum of 1410 operations to obtain the worst-case secret password. The budget of 1500 operations and 80% success is insufficient to overcome the issue with secret characters being dropped randomly.

Interactions with user profile information was audited to ensure that it was not involved in the login process in some other way to rule out alternate side channels.

User-controlled input related to the password was reviewed using taint.

- Taint and the PCG builder were also used to determine that key variables were always null, rendering the dead code mentioned in the report.
- Custom queries, taint, and the PCGBuilder were used to review user profile information starting from Voter.voterTraits"

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.10.3 Question 034 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"There is a user-controlled back door by which the INCLUSION_THRESHOLD can be used to control whether candidate names are included in the results. This also has an effect on compression of a summary message, making it a possible side channel in space. However, the INCLUSION_THRESHOLD is an integer, whereas the calculation for percentage is [0,1]. Therefore, candidates can potentially be distinguished based on whether they have 100%, but not within 1%."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable. The team identified the same weak side channel as several other teams and correctly noted that it was not sufficiently strong to result in a vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.10.4 Question 037 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"User-controlled input can reach two places of interest: LoginManager has two methods which process the password: handleGet and handleParcel.

1. In handleGet, the codeExchangeController is always null, rendering any potential side channel in the 'then' branch unreachable.
2. In handleParcel, any potential side channel in DESHelper.pullEncryptedString() is unreachable because passwordCode is always null. The password comparison occurs in TokenedPasswordChecker.processPassword(). The result of the password comparison is a token which is incorporated into error output, making it a potential side channel in space. However, processPassword() includes characters from the secret password with 50% probability, making attacks infeasible within budget. Worst-case secret passwords have 15 characters from an alphabet of 94 characters, which implies a minimum of 1410 operations to obtain the worst-case secret password. The budget of 1500 operations and 80% success is insufficient to overcome the issue with secret characters being dropped randomly.

Interactions with user profile information was audited to ensure that it was not involved in the login process in some other way to rule out alternate side channels.

- User-controlled input related to the password was reviewed using taint.
- Taint and the PCG builder were also used to determine that key variables were always null, rendering the dead code mentioned in the report.
- Custom queries, taint, and the PCGBuilder were used to review user profile information starting from Voter.voterTraits"

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.10.5 Question 038 (AC Time, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"A review of all loops, lambdas related to loops, and recursion and was performed.

There is expensive code which processes the registration code, RegistrationGate.check(), but it cannot be exploited within the input budget. […] The cost of logisticMapping() is roughly 2m, where mis related to the difference between n and the last cached value. To drive up the difference, MAX_N must be increased. This is accomplished whenever the value of n is exactly MAX_N. MAX_N only increases by 1 each time. Because the mod 7 check happens before the expensive operation, it is necessary to send a valid registration code to trigger the costly

operation. This implies that the final registration code has to be divisible by 7, such as 1001. Because the final registration key has to be repeated to pass the isInBounds() check, the minimum attack is 8 registration codes. However, empirical measurements show that 8 registration codes is slightly less than 300 seconds. Further, login plus 8 registration codes exceeds the input budget of 4096 bytes, as measured in terms of TCP length fields. We observed that SSL encryption overhead per registration code seems to be the dominant factor in preventing the attack from succeeding within budget.

[…] There is a potential infinite loop in fetchResponse(), and recursion in the visitors for saving a Response. Response.query could be made to point to itself by sending a ballot update where the answer points to itself. […] However, the apparent infinite loop in fetchResponse()is a guard against this. It replaces the user input with the existing parent Query of the Response prior to setting Response.query. The Dashboard identifies potential call graph recursion in the app, which identified the potential recursion in the Response/Query visitors.

Checking whether the Response/Query infinite recursion was feasible when performed using custom queries. In order for it to be feasible, an instance of a Response had to reach Response.query. Therefore, the taint graph between the identity (this) in Response.assignQuery() and the query field was reviewed for places where the type was known to be a Response, revealing the infinite loop in fetchResponse() that turned out to be the necessary guard condition.

[…] Finally, the Loop Call Graph and Lambda Loop Catalog were reviewed to rule out other sources of AC vulnerabilities."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Thanks to Northeastern (Fish Wang), who implemented a script which holds the SSL connection open, thereby avoiding the overhead which was pushing the input bytes over budget for our previous attempts. The script is included alongside the report.In simplevote_3, the "n" component of registration keys must be divisible by 7 to trigger expensive computation. The minimumsequence to trigger the vulnerability is n = 995 to 1001, followed by 1001 (8 total). The script contains an example sequence.

We observed a runtime of 7 minutes 17 seconds, 2893 input bytes, well over the target of 5 minutes (300 seconds) and inside the input budget of 4096 bytes.

Measurements from our collaborators:

Colorado: 7m 17s

Vanderbilt: 7m 23s

Utah: 7m 20s

GrammaTech: 7m 18s

Side note: the previous report mentions a sequence of 9 registration keys which was known to not be a feasible attack for simplevote_3 because 1002 is not divisible by 7; this was likely a copy/paste error from Question 30. To increase the runtime of the attack, we would need to send

15 registration keys, but when tested with the new script this required 4811 input bytes, rendering such an attack infeasible."

Live-Collaborative Evaluation:

The unintended vulnerability reported following the live-collaborative engagement was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.11 SimpleVote 4

A.6.3.2.11.1 Question 023 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"There is expensive code which processes the registration code, SanctionAuthenticator.check(), but it cannot leak the registration code in time. […] Unlike other variants of SimpleVote, this version neither recurses nor loops over the expensive operation. The other variants repeat the expensive processing n times, which is how the registration key is leaked in time (determined by observation of the server response time).

[…] Further, StraightSelectionServiceActual.confirmSanctionKey() checks that the entered registration key is associated with the voter. Viewing the ballot requires entry of the registration key, and the registration key must match the voter who is currently logged in, which implies that the target voter's password must also be obtained. Per Question 41, the password cannot be obtained in time. If the attack were otherwise enabled, then the side channel strength would depend on how distinct the processing time is for each value of n. Timings are roughly constant, independent of n, due to internal caching of intermediate computations.

An audit of the user-controlled input using taint, especially password and registration code, were used. In addition, custom queries were written to search for and audit uses of BigDecimal, which lead to an audit of SanctionAuthenticator."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.11.2 Question 030 (AC Time, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"A review of all loops, lambdas related to loops, and recursion and was performed.

There is expensive code which processes the registration code, SanctionAuthenticator.check(), but it cannot be exploited within the input budget. […] Registration keys are 12 characters; when

unscrambled the last 6 characters form a number n, which (to be a valid registration key) must be divisible by 7, and the first 6 characters have to match a function of n. The cost of chaoticOperator is roughly 2m, where mis related to the difference between n and the last cached value. To drive up the difference, MAX_N must be increased. This is accomplished whenever the value of nis exactly MAX_N. MAX_N only increases by 1 each time. Because the mod 7 check happens after the expensive operation, it is not necessary to send a valid registration code. However, empirical measurements show that it still takes 8 or more registration codes to get close to exceeding 300 seconds. Login plus 8 registration codes exceeds the input budget of 4096 bytes, as measured in terms of TCP length fields. We observed that SSL encryption overhead per registration code seems to be the dominant factor in preventing the attack from succeeding within budget.

[…] There is a potential infinite loop in obtainResponse(), and recursion in the visitors for saving a Response. Response.query could be made to point to itself by sending a ballot update where the answer points to itself. However, the apparent infinite loop in obtainResponse() is a guard against this. It replaces the user input with the existing parent Query of the Response prior to setting Response.query. […] The Dashboard identifies potential call graph recursion in the app, which identified the potential recursion in the Response/Queryvisitors.

Checking whether the Response/Query infinite recursion was feasible when performed using custom queries. In order for it to be feasible, an instance of a Response had to reach Response.query. Therefore, the taint graph between the identity (this) in Response.fixQuery() and the query field was reviewed for places where the type was known to be a Response, revealing the infinite loop in obtainResponse that turned out to be the necessary guard condition.

Custom queries were used to identify user-controlled input based on HttpExchange, and taint was used to follow input to potential vulnerable code. In addition, custom queries were written to search for and audit uses of BigDecimal, which lead to an audit of SanctionAuthenticator.

Finally, the Loop Call Graph and Lambda Loop Catalog were reviewed to rule out other sources of AC vulnerabilities."

Take-Home Evaluation:

The challenge contains an intended vulnerability in the processing of registration keys by which an attacker can trigger the cache use to speed up processing to be wiped.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Thanks to Northeastern (Fish Wang), who implemented a script which holds the SSL connection open, thereby avoiding the overhead which was pushing the input bytes over budget for our previous attempts. The script is included alongside the report. In simplevote_4, the "n" component of registration keys does not have to be divisible by 7 to trigger expensive computation. However, more registration keys are required to trigger sufficient runtime as compared to simplevote_3. The minimum sequence to trigger the vulnerability is n = 995 to 1002, followed by 1002 (9 total). The script contains an example sequence. Input bytes: 3167 (vs budget 4096)Runtime: over 10 minutes (vs budget 5 minutes)

Vulnerability 2

Thanks to Colorado and GrammaTech, who both discovered a vulnerability we missed. The attack can be replicated by sending three registration keys, all corresponding to n = 995. Compared to vulnerability 1 above, the input bytes will less because it takes fewer registration keys, and therefore is in budget. We observed a runtime in excess of 7 minutes, over the 5-minute target."

Live-Collaborative Evaluation:

Following collaboration, Iowa reported the intended vulnerability an unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.6.3.2.11.3 Question 039 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"There is a user-controlled back door by which the INCLUSION_THRESHOLD can be used to control whether candidate names are included in the results, down to 1% increments. This has an effect on the size of the summary message, making it a possible side channel in space. However, compression is initialized using the entire summary message and the user-controlled backdoor name (stored in static field data). This leads to chaotic compressed result message sizes, and prevents a binary search on the candidate's percentage. This was confirmed empirically. The app was instrumented to print out the size of the compressed message in bytes, and tested with the candidate "Rocky De La Fuente" for percentages ranging from 0 to 100 in increments of 10. If binary search were possible for "Rocky", the message size would stop changing around 40, but it continued to change between 0 and 80

Custom queries were used to find peer-to-peer message passing, starting from interaction with google protocol buffers. Taint was used to review back door operation and interaction with message construction."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.11.4 Question 041 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"User-controlled input can reach two places of interest: LoginGuide has two methods which process the password: (1) handleDeliveryand (2) handlePull.

1. In handlePull, the keyExchangeServer is always null, rendering any potential side channel in the 'then' branch unreachable.
2. In handleDelivery, any potential side channel in DESHelper.pullEncryptedString() is unreachable because passwordKey is always null. The password comparison occurs in

passwordsEqual(); the comparison includes all characters, and there is little or no difference for matching vs not matching.

Interactions with user profile information was audited to ensure that it was not involved in the login process in some other way to rule out alternate side channels.

User-controlled input related to the password was reviewed using taint.

- Taint and the PCG builder were also used to determine that key variables were always null, rendering the dead code mentioned in the report.
- Custom queries, taint, and the PCGBuilder were used to review user profile information starting from Voter.voterTraits."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.12 STAC Coin

A.6.3.2.12.1 Question 007 (AC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"We discovered an algorithmic complexity vulnerability in the App that can be triggered using a malicious peer. We were able to verify that the attack is within the budget and works with a greater probability than specified.

An attacker can cause the benign peer to send requests to a malicious peer that never returns a request to the benign peer. Because the request from the benign client does include a timeout setting, this hangs the Thread that made the request. If the attacker does this enough times it can hang all of the available threads on the benign peer. After this is done, the benign peer is unable to respond to the benign request. […]

As per the Dashboard, the entry point of this App is Stac.main. Stac.main handles all the requests in this App and redirects them to various methods for processing. Any code segment that allows the attacker to interact with the victim is in this method. Hence, we focused our attention on this method. Stac.main uses lambda expressions to redirect the requests to various handlers. It shows that there are three lambda expressions which are of particular interest because the attacker can control the operations defined by them. We suspected that these requests can be used to craft an AC attack.

While investigating, we found that one particular request -POST request to "/wallet/spend" interesting. The attacker can send a request of this type -which translates to sending a STACCoin to another peer. The peer responds by checking whether it's a valid transaction and requesting a

transaction to start mining. If the attacker does not respond back with the mining request, and if there is no timeout on the benign peer's side, then that particular thread waits

forever. Our analyzer did not find a timeout set operation and hence it does not show up in the PCG. Thus, there is nothing preventing the theorized attack."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. The potential unintended vulnerability reported by Iowa is dependent on the attacker having the target user's wallet password, an assumption not provided by the challenge question or through a clarification following the start of the engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We originally answered this question with "Yes" and created an exploit. During the collaboration we shared our exploit with other teams. Two teams, one being us the other being GrammaTech, tried to recreate the exploit on the reference platform during the collaboration. While recreating the exploit, we noticed that we were making an assumption about the question. This assumption was that the Benign User would initiate 400 separate spend requests and then they would call the GET /balance/me on their server. The server would not respond because all the threads would be taken up handling the 400 requests to the malicious attacker and hang their interaction. This assumption was found to be not in scope and would not be guaranteed to produce a vulnerability depending on the Benign User's discretion.

We have another hypothesis which we did not have time to fully test. We suspect there is a vulnerability where the attacker can create a malicious user similar to the above implementation and have the attacker send the 400 spend requests and get the other server to block on an response packet. We think the response that we can control in connection is the GET /blockchain/length request which is sent to the malicious peer after the new transaction is created and sent to the Benign Peer. We were unable to test this possible exploit and would need further analysis to find evidence of a vulnerability."

Live-Collaborative Evaluation:

The challenge does not contain an intended vulnerability. Following the collaborative engagement, Iowa reported another potential vulnerability it is unclear from the information provided if it is viable, but the team correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.12.2 Question 008 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)*

Take-Home Response:

"The question is whether the attacker can send an input request which forces the app to store a file with size exceeding the resource usage limit. Thus, we looked for all places in the source code where a file write operation is performed. Using Type and CallSite catalog we found that there are 651 File objects defined in the App and imported libraries. Also, there are 9792 references to the File objects. As we are interested only in the write operations on these references, we used the Type and CallSite catalog to find the callsites performing write

operations using the File objects. There are only 2 such callsites within the App. These callsites are in the methods Persistence.writeWalletToDisk and HTTPUtil.sendHTTPSPost. Since, these are the only two file write operations within the App. We focused on these two methods.

To understand the control structure of these two methods with regard to the File write operations, we created the PCG corresponding to these two methods. As the callsites to the File write operations could be in a try-catch block we also included the exceptional control flow edges in the PCG. The PCG reveals that neither of the callsites are governed by loops or interesting branches. Further, the File write operation in the sendHTTPSPost writes contents of a local variable. Hence, the local copy will be renewed each time the method is called. So, it is not likely to exceed the usagelimit. The callsite in the method writeWalletToDisk operates on a field. It reads the contents of the field and writes it to a file. The field is a static object, and if the size of the object exceeds the usage limit, then the written file will also exceed the usage limit. Thus, we analyzed that static object. The static object is of type Wallet that has two fields of types: BlockChain and Map. Thus, if the attacker wants to exceed the budget, he should either extend the length of the BlockChain variable or inflate the Map field or both.

Hypothesis 1: The BlockChain in Wallet can be inflated by the attacker to exceed the usage limit. We used the Type and CallSite catalog to find any loops that inflate either the BlockChain or the Map. We found 17 such loops. In order to understand the interprocedural control flow between them, we induced the call edges between the methods containing these loops. This graph reveals that the methods SCHash.padBitstringWithLeadingZerosToLength and SCHash.getFirstPrimes are isolated. Hence, they are not as interesting. Wallet.spendforms the root of a different component of the graph. We tested this call chain dynamically and it did not exceed the usage limit. […] In order to verify that the BlockChain does exceed beyond the usage limit to cause the vulnerability, we switched to dynamic analysis. We verified that the attacker can indeed trigger the loop in the Miner.run. But our analysis did not reveal the existence of an input which can exceed the resource usage limit within the given budget. Therefore, we refute Hypothesis 1.

Hypothesis 2: The Map in Wallet can be inflated by the attacker to exceed the usage limit. From our analysis, we did not find a loop that inflates the Map in Wallet. But, there is a possibility that the attacker can call the methods writing to the Map repeatedly and make it exceed the usage limit. Using Taint Analysis, we found there are only two methods that can add to the Map: Wallet.addKeyPair and Wallet.generateNewKeyPair. We verified that the attacker can trigger both of them. But, we did not find an input that can inflate the Map beyond the usage limit using these two methods within given operational budget. Therefore, refuting Hypothesis 2”

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“During the collaboration, team GrammaTech described an attack which we had missed. This attack works under an assumption. We first describe the attack and then the assumption.

Attack

1)The attacker builds a large blockchain on his side without interacting with the victim.

2)After the blockchain is built, then the attacker creates a transaction corresponding to this large blockchain. Then he sends a request to the victim for mining the created transaction.

3)The victim then mines the transaction and tries to update his blockchain . This happens in Miner.announceBlock method.

4)Miner.announceBlock calls the method Miner.tryToUpdateFromPeer. This method contains the vulnerable loop.

5)This expands the blockchain on victim's side. Then the victim tries to write it to a log file (by calling Persistence.writeWalletToDisk).

6)Because the blockchain is now very large, the file written is also big and exceeds the budget.

Assumption: The assumption made by team GrammaTech is as follows:

In order to create a large blockchain, the attacker needs to create many transactions using the spend request and mining the transaction. This needs a mining request to be sent from the attacker to the victim. GrammaTech argued that this mining request can be done without it being counted against the budget. This is not counted against the budget because it is done locally. It still needs to send a response back to the victim even if the mining is done locally.

GrammaTech assumes that since the budget is only about the requests sent by the attacker and not the responses this attack is within scope. Hence, under the assumption stated, we change our answer to YES."

Live-Collaborative Evaluation:

Iowa noted in their response, that the response of "yes" is dependent on an interpretation of the budget. The input budget applies to the total bytes sent from the attacker to the device with the application under attack. We note that this misinterpretation that bytes sent in response did not count towards the budget was the motivation for the response, therefore this response is changed to a "No"

Post-Engagement Analysis Ruling: Correct

A.6.3.2.13 Swappigans

A.6.3.2.13.1 Question 002 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"[…] one possibility for the attacker is to make the server crash or hang in order to stop the server from processing any subsequent benign requests. We analyzed the App to confirm or refute this hypothesis. It is very likely that this App contains a loop which throws an unhandled exception or performs a time-intensive operation (e.g., iterates for a long amount of time). In both cases, the server is busy processing some request and the AC time vulnerability can be triggered. Using Loop Catalog, we found 11 loops in the App. 3 of them are not controlled by the attacker (not user-input controlled). One of these loops is used to list all the available items for the purchase. While the input (in this case is the request to list all items) reaches the loop (in the method ListItems.getItemList), it is not very interesting as it is not doing any expensive operation. That leaves us with 7 loops.

[…] One of the components deals with construction of a SwappigansItemGroup object. This object is used in methods SwappigansItemMatcher.calc,

SwappigansItemMatcher.apxSumRecursion and, SwappigansItemMatcher.mergeTrimRemGreater. While the object created by this component can lead to the vulnerability when used, the loops in this component which create the object are not resource intensive. Hence, they cannot cause the vulnerability by themselves. In the other component, one call chain leads to addition of new items (AddItem.getResponse calls AddItem.priceToInt). The attacker can control these operations by adding new items to his inventory. But it is not particularly interesting code structure on its own as adding a new item is not time consuming. The final call chain is what causes the AC time vulnerability. It deals with the "purchase item" request. The App processes the "purchase item" request as follows: It retrieves the price of the item being purchased and the list of items in the inventory of the purchaser. Then, using a knapsack-like algorithm it attempts to find the best set of items owned by the purchaser which match the price of the item being purchased. This algorithm is implemented in three methods -SwappigansItemMatcher.calc, SwappigansItemMatcher.apxSumRecursionand, SwappigansItemMatcher.mergeTrimRemGreater.

SwappigansItemMatcher.mergeTrimRemGreater contains two loops and SwappigansItemMatcher.apxSumRecursion is a recursive method. This raised our suspicion.

Using the PCG, we discovered that the runtime of the algorithm is dependent on the number of items in the inventory of the purchaser. Given that the attacker can control the number of items in the inventory using the AddItem operation, we hypothesized that the attacker can make the server wait long enough using the "purchase item" request. It is a triggerable vulnerability and within the operational budget. For triggering the vulnerability, the server needs to either hang or crash. We discovered that as the number of items increases, it takes longer for the server to respond. Therefore, the attacker requests to add more items to reach the permissible maximum number of items allowed within the operational budget, and he will be able to successfully crash the server. After the server crashes, no other thread will get a response back from the server. Hence, exceeding the given resource usage limit and triggering the AC Time vulnerability."

Take-Home Evaluation:

The challenge contains an intended vulnerability, but it requires the attacker to bypass the input guard on the maximum price before using a number of items to exceed the resource usage limit while purchase an item of price $>$ \$1000. Iowa state does not provide the full details of the unintended vulnerability, but they point to the same unintended vulnerability flagged by several other teams – that the resource usage limit can be exceeded when purchasing an item with a price of $\leq$ \$1000.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"A better attack strategy to exploit the same code segments was learnt while collaborating with Colorado and Vanderbilt. The vulnerable code is the same but the worst-case input/steps for the attack are as follows

1)Attacker creates a user

2)Attacker adds items in increasing/decreasing order of the price. If the items are added in increasing order, double the price of the item being added must be double the previous item's

price. If the items are added in decreasing order, price of the item being added must be double of previous item's price

3)Attacker creates another user and attempts to purchase the highest priced item

4)Then the benign user tries to log in.

5)The purchase request of the attacker takes a long time to complete, the benign user is denied the service."

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.13.2 Question 020 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The attacker monitors the network traffic passively and waits for the target user to send a valid request like a login request to a server. This triggers generation of saltString in server. Within DELTA ms, the attacker uses the same algorithm to generate saltString on his computer. The vulnerability is that the same saltString is generated every 1000ms (1 second). The attacker uses the saltString and username to generate the session key using the same algorithm used by the server. Having the username and session key allows the attacker to impersonate the target user, hence making a purchase.

[…] For this attack, we assume that the System time on the attacker and server computer is close enough that if it is executed within DELTA ms, then it returns the same seed in this computation, Random.setSeed(System.currentTimeMillis() / 1000);

The session key is created in SessionManger.getSessionToken(String username) method. Taint analysis reveals that the this session key is based on the username and a Cipher based on "key" which is based on saltStr. Backward taint on the saltString reveals that it is randomly generated and the seed is set as follows: Random.setSeed(System.currentTimeMillis() / 1000);

Using Dynamic Analysis, we confirmed that the seed remains the same if the two requests to Random.setSeed are sent within DELTA ms. Since the argument to Random.setSeed is System.currentTimeMillis() / 1000, it means that a new seed is generated every 1 second (1000 ms). Therefore, depending on the time when the server generates it's key, the attacker will have anywhere between few milliseconds to 1000 ms to generate the same key.

Hypothesis: If the attacker can generate the seed within DELTA ms of when the server generates this, then the attacker can get the saltString. Since the attacker also has the username, the attacker will be able to obtain the sessionkey for that username. When does the server generate SaltString? The reverse call graph on SessionTokenManger.getSaltString(int) method shows that this gets called only during Registerrequests and class Initialization. The class initialization will happen when any public fields or methods are called for the first time. […]

Depending on when the server first generates the session key, the attacker will have a window of anywhere between few milliseconds to close to the full 1000 ms to generate the same key. At least in 50% of the cases, the attacker will have sufficient time (more than 500 ms) to generate

the same session key. This attack scenario fits within the given operational budget and can happen with 50% for probability of success."

Take-Home Evaluation:

Iowa flagged the same unintended out-of-scope vulnerability as several other teams. A key assumption of this vulnerability is the attacker knowing when the server was started. In future challenges we will work to clarify that the attacker does not know when the server was started.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"At the collaboration, all the teams agree that this attack can work. From the discussion it became clear that because of the faulty algorithm for saltString generation, no other information is required to carry out this attack. Vanderbilt brought to our attention that the session key is susceptible to a cryptographic attack. Based on the clarification provided by the White Team at the time of collaboration, we are to report presence of a vulnerability which can produce the described effect. Hence, we are sticking to our original answer. But we do note that the side channel is vacuously true, its observables are empty."

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.13.3 Question 031 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Our analysis did not reveal a side channel strong enough to leak the secret. Our analysis led to the class ItemPurchase which handles the purchase operation in this app. This particular app uses the field itemId to identify which item was purchased. This is the secret for this question. […] Since the attacker is neither allowed to make the purchase, nor is he allowed to make changes to the database, the passive operation of observing the network traffic generated by the victim becomes important. As the victim is going to take a single action, the observable is the time it takes to process the request to make a purchase. Our loop reachability analysis helped us to find the loops which are reachable from the entry point for ItemPurchase. We found that the ItemPurchase.getResponse is the method which processes the purchase.

there is also a recursive method apxSumRecursion. Since, these are the only artifacts interacting with the secret, we focused our analysis on them. Using the information gathered using our static analysis we tried to formulate an attack. The description of the app gives us the following information -It is a barter app. To purchase an item, a list of items must be offered which have a combined price greater equals the item being purchased. This list is computed by the application. From our analysis, we conclude that apxSumRecursion must be computing the same. This particular algorithm is dependent only on the price of the item being purchased. Hence, we hypothesized that based on the prices of the items, processing of the purchase request would take different amounts of time and the items would be distributed into buckets based on these timings.

[…] We logged in as each user and measured the time taken to the complete purchase of each item. We have attached, the CSV file containing the results of the experiment. From the CSV file, we concluded there is no deterministic difference between the items based on the time

required to purchase them. Hence, there are no buckets formed and thus we concluded the absence of the side-channel vulnerability in time."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.3.2.14 Tollbooth

A.6.3.2.14.1 Question 011 (AC Time, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"We found a way to cause the delay in the processing of the benign request exceeding the usage limit. But, we can statistically prove that the probability of success of the attack is about 25% as opposed to the stipulated 50%. Hence, we concluded the app does not contain an AC Time vulnerability.

Based on the description, we identified that the benign request is sent to a leaf peer. Using the taint, we identified that the packet is added to a queue in the leaf peer for processing. The way the packet is processed is as follows,

1) packets are read from the queue of the leaf peer. (in the method TollBoothWorker.run)
2) Every 500 ms, one packet is chosen randomly out of the packets read and sent to the root manager. (TransponderDevice.run introduces this delay)
3) Root manager processes the packet and sends a response to the leaf peer. (in the method TollBoothRootWorker.handlePacket)
4) Leaf peer then forwards the response packet to the benign user. (in the method TollBoothWorker.run)

Hypothesis 1: It is possible to manipulate the random selection of the packets such that the benign packet is not selected before the resource usage limit is exceeded. We used the branch catalog from our Dashboard to identify the interesting branches. The highlighted branch in the above code segment is interesting as it depends upon a random number and has a peculiar functionality. Specifically, based on the decision of this branch either a packet is sent to the root manager or not. Additionally, a random number between 0 and the number of transponders (knownTransponders) selects which packet is sent to the root manager. For each iteration, if the benign packet is not selected, a delay of 500 ms is caused.

In order to exceed the usage limit, the attacker needs to cause the loop to iterate at least 500s/500ms = 1000 times. The selection of the packet is dependent on two random numbers (random1and random2). The first one (random1) is always between 0 and 100. The other (random2) is dependent upon the number of transponders in the App and is used as an index to select a packet from the knownTransponders set. Because of the dependency of random2on the

number of transponders in the App, we looked into ways to control this variable. From the description, we figured out that using the "Register" request the attacker can add new transponders to the App and control the number of transponders. Based on the code shown above, we concluded that with more transponders in the App, it is less likely for the benign packet to be selected. Hence, the attacker should aim to increase the number of transponders in order to maximize the likelihood of not selecting the benign packet. Thus triggering the AC vulnerability. Using dynamic analysis, we figured that the attacker can add a maximum of 174 transponders to the App within the given budget.

[…] First, the transponders passing through a leaf peer are assigned a unique id chosen randomly Next, the leaf peer generates a charge_transponder request to the root manager to process. The benign packet is one out of the 178 possible charge_transponder requests that can be generated. […] the likelihood of benign packet being selected in each iteration is $\frac{1}{712}$. As the events are independent, the probability of the benign packet not being selected in 1000 consecutive iterations is: P(benign packet NOT being selected) $= \left(1 - \frac{1}{712}\right)^{1000} \approx 24.52\%$ As the probability of success is expected to be 50%, we conclude that this hypothesis will not work.

Hypothesis 2: Benign packet is not processed by the root manager within the usage limit. Root manager simply reads the packet from the queue and generates a response packet. The only way to delay this would be to increase the length of the queue. As we said before, the maximum number of packets we can add to the queue are 174. We dynamically verified that even after adding the maximum possible packets to the queue, the attacker cannot delay the benign request by more than 500s (usage limit)."

Take-Home Evaluation:

Iowa missed the intended vulnerability by which an attacker can submit a malformed packet to cause the root tollbooth node to fail to respond to a charge request packet. Iowa's response could indicate the team mis-understanding the construct for the challenge question. The leaf peer is simulating transponder traffic and is not what is under attack. The code running on serverNuc (the root manager) is under attack. The construct is that charge_transponder requests are sent from the leaf peer to the root node and the attacker's goal is to delay the charge_transponder requests that occur.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We are changing our answer from No to Yes due to information brought to us from the team Grammatech. Their proposed vulnerability was to send a malicious packet from a custom client to crash the root node's listener for handling requests from the queue. At the meeting in the collaboration they did not have an exploit to show, but we were able to create an exploit from experimentation after the meeting. The exploit comes from creating a malicious client that causes the root node to crash its listening-for-request threads. The steps to reproduce are the following:

1.Start the root node on the serverNUC

2.Start a leaf node on clientNUC

3.Start the malicious client on the masterNUC

a.Malicious client is defined as a client with the CarPassedPacket class to have a modified constructor of a null destination supplied to its parent

4.Wait until the malicious client hits the carPassed method under the TollBoothWorker class

5.See that the serverNUC is still running the root node but the listening thread has crashed

There is a known bug in with Java's default implementation of Random where the numbers generated leak the state of the Linear Congruential Generator. This allows the attacker to adjust their position in the set of known Transponders (which is a sorted Tree Set) so the attacker can position the value of a transponder id into a spot that could cause the vulnerability."

Live-Collaborative Evaluation:

The exploit provided by GrammaTech was confirmed as a valid way to crash the thread that processes the benign request to be delayed

Post-Engagement Analysis Ruling: Correct

A.6.3.2.14.2 Question 024 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Based on our analysis, we are able to discover a potential side channel in the App but it is not strong enough to trigger the vulnerability specified in the description with the given operational budget. The side channel we seek should leak information that allows the attacker to transfer funds from a benign Transponder Id to his Transponder Id. As the secret is not well defined in the question, we must first find the operation that transfers funds between two transponders, understand it and, look for clues on how an attacker can exploit the App to transfer funds to his account from benign user account. […]

The analysis of the add_value request reveals that it incorporates the following 4 parameters:

1. Transponder Id: The Id of the transponder which is transferring the money.
2. Value: Amount of money to be added to the transponder balance.
3. Transfer Id: The Id for the transponder receiving the money.
4. Transfer Value: Amount of money to be transferred to Transfer ID account.

[…] for the attacker to exploit this App and transfer funds from a benign user account to his account, the attacker needs to issue add_value request with the following values:

1. Transponder ID: The Id of any benign user.
2. Value: value should be a negative one, so the benign user can be tricked to transfer money to the specified Transfer Id.
3. Transfer ID: The attacker Id so he can receive money.
4. Transfer Value: Amount of money to be transferred to Transfer ID account.

All parameters to issue the add_value attack request are known except for the Transponder ID corresponding to a valid benign user. In this case, the benign user Id is our secret. Once known, the attacker can issue the attack request to transfer funds from a benign user account to his account.

Hypothesis: There exists a side channel is space and time in the App, that will enable the attacker to observe difference in timings and sizes of the network traffic to reveal the Transponder ID of a

benign user and used it to issue add_value request to transfer funds from benign user account to his account. […] Knowing that the 4 default transponders are all leaf peers, we focused our attention on the method TransponderDevice.run. This method is called once the TransponderDevice instance is constructed in TollBooth.main method. […]

The charge_transponder request can be passively observed by the attacker in the network traffic, So probably there exist some correlation between the size and timing of the charge_transponder packet and the randomly selected Transponder Id in TransponderDevice.run method. To verify this claim, we used our dynamic analysis to capture the network traffic between the leaf peer and the root peer. We found that all charge_transponder request packets have the same size and there is no substantial difference in timings and sizes of induced response from the root peet in reply to charge_transponder packets. Hence, we concluded that the attacker cannot reveal the secret (any of the benign Transponder Ids) using passive operations.

As the attacker cannot reveal the secret (any of the benign Transponder Ids) using active and passive operations, we refute our hypothesis and conclude that the SC vulnerability in space and time described in the question is not present in the App."

Take-Home Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Iowa correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4 Northeastern University

A.6.4.1 Northeastern Overview

Northeastern answered the second most questions during the take-home engagement (41), but achieved the second lowest accuracy of 61%. Northeastern had the lowest TNR of 69% and the fourth highest TPR of the R&D Teams, 47%, in the segmentation of answered questions. The team had the most difficulty with AC Space and SC Space questions with a 38% accuracy and a 50% accuracy respectively. The team had the most success with SC Time/Space questions with a 100% accuracy.

In the segmentation of all questions (unanswered questions assessed as incorrect), Northeastern achieved the fourth highest R&D Team accuracy of 58%, with the R&D Teams all outperformed by the control team. The team had the fourth highest R&D Team TPR (44%) and TNR (67%) in the segmentation of all questions. In our assessment of the reasons for incorrect take-home responses, Northeastern's incorrect SC responses were due to improper strength analysis, leading the team to incorrectly declare challenge programs vulnerable. The team's incorrect AC responses were due missing the complexity within the challenge program, leading the team to incorrectly declare the challenge program non-vulnerable.

During the live-collaborative engagement, all R&D Teams achieved between a 97% and a 100% accuracy. Northeastern tied with Vanderbilt and GrammaTech with a 98% accuracy. Collectively, the R&D Teams identified 3 unintended vulnerabilities during the live collaborative engagement – Railyard Question 013, Class Scheduler Question 017, and SimpleVote Question 038. Railyard Question 013 and Class Scheduler Question 017 came out of a collaborative effort from all teams. The SimpleVote Question 038 unintended vulnerability was confirmed with a clever exploit by Northeastern. The only question that the R&D Teams collectively got incorrect was the intended SC Space vulnerability in Case DB Question 019.

A.6.4.1.1 Take-Home Engagement Scores

Table A-129: Engagement 7 NEU Take Home Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	4	50
SC Time	12	8	67
SC Space/ Time	4	4	100
AC in Space	8	3	38
AC in Time	9	6	67
Total	**41**	**25**	**61**

Table A-130: Engagement 6 NEU Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	15	7	47
Not Vulnerable	26	18	69
Yes Answer	16	7	44
No Answer	25	18	72

A.6.4.1.2 Live-Collaborative Engagement Scores

Table A-131: Engagement 6 NEU Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	7	88
SC Time	12	12	100
SC Space/ Time	4	4	100
AC in Space	8	8	100
AC in Time	9	9	100
Total	**41**	**40**	**98**

Table A-132: Engagement 6 NEU Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	15	14	93
Not Vulnerable	26	26	100
Yes Answer	15	15	100
No Answer	26	25	96

A.6.4.2 Northeastern Specific Responses

A.6.4.2.1 BattleBoats 3

A.6.4.2.1.1 Question 022 (AC Space, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"None of the operations available can cause program to exceed memory limits. We checked loops reported by our static analysis as well as manually checked the program. The loop for calculating the shot was a bit suspicious but has hardcoded limits to prevent it from looping too much."

Take-Home Evaluation:

Northeastern did not identify the intended vulnerability by which a hash table can be used to exceed the resource usage limit during shot computation.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Colorado as well as GrammaTech found shots that took a large amount of memory and exceeded the limit. This was surprising to us because there was a fixed limit on the number of iterations of the shot calculation. We suspect that this is an issue related to creating many BigDecimal instances in the number of iterations we were given. No one really understood how the code was leading to this AC originally, but Iowa State eventually found that the square root algorithm might be the problem since it converts doubles to highly precise BigDecimals."

Live-Collaborative Evaluation:

Through collaboration, Northeastern was able to correctly identify the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.1.2 Question 026 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"We believe there is no reliable side channel in space. The size of the responses depends on the ship that is hit. However, this does not help in predicting the ship locations."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.1.3 Question 040 (SC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"In OceanScreen the method pullOceanSquares only performs a significant amount of computation if it's on the board. This timing difference can be used to determine if your shots are on the board, and therefore where your cannon is. This is enough to let you find all the ship locations because you can check 4 squares at once and you know if you are hitting the board. Without the side channel to determine where your cannon is you might have to check a lot more locations."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. We don't believe that the observed time difference is sufficiently strong to result in a vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In our tests, we consistently observed timing differences between shooting just on the board and just off the board. However, other teams did not observe a timing difference. One team instrumented the artifact and we measured it repeatedly. Unfortunately, we did not see the expected difference where a shot outside the board is faster."

Live-Collaborative Evaluation:

Northeastern was able to correctly change their answer after consulting with their collaborative partners.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.1.4 Question 043 (SC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"In OceanScreen the function pullOceanSquares only performs a significant amount of computation if it's on the board. This timing difference can be used to determine if your shots are on the board, and therefore where your cannon is."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability. We don't believe that the observed time difference is sufficiently strong to result in a vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Though the justification was the same as the Take-Home, this team correctly judged that the potential find was invalid.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.2 BraidIt 3

A.6.4.2.2.1 Question 004 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“The braidit server logs information to staclog/stac.log. If an attacker plays the game by making a guess, selecting a braid, but then submitting their own braid, they can control the content of the log. If an attacker constructs a braid with many (xY; yX) pairs for many different letters in the alphabet, then the log file grows very large. A 1.6KB braid can generate a ~25MB log file, so it seems very likely that 2KB worth of bandwidth could generate a log file outside the stated limit of 25MB.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The challenge program contains an unintended vulnerability identified by Iowa and Two Six Labs. The justification provided by Northeastern was insufficient to confirm an unintended vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Braid minimization occurs in a loop where a logfile at staclog/stac.logis appended to. A malicious braid can easily be generated that will cause enough iterations of the loop to execute such that the attack success threshold is satisfied. One obstacle to creating this condition at the victim’s braidit instance is that the random braids generated by the victim to be modified by the attacker will never possess the structure required to carry out the attack. However, a malicious client can carry out a modified version of the network protocol where, instead of modifying the braids submitted by the victim, they can instead submit their own braids back to the victim. No error was observed when implementing this modification, and the resulting PoC demonstrates the presence of this vulnerability in the target artifact.”

Live-Collaborative Evaluation:

The live-collaborative response makes it difficult to confirm the vulnerability reported by the team; however, given the collaboration with other teams that identified a similar vulnerability and provided more specificity as to the cause of vulnerability within the code/provided an exploit this response is assessed as correct.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.2.2 Question 027 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“We carefully analyzed the program code and studied its responses by monitoring the size of packets sent from the opponent to the adversary where the opponent selects a different braid in each game. However, the size of the packets sent by the server do not depend on the opponent’s

selected braid. In fact, in all games we observed the same packet sizes being sent from the opponent to the adversary."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.2.3 Question 029 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"The application appears to implement RSA using Montgomery Modular Multiplication, which can be susceptible to side-channel attacks if incorrectly implemented. In the decryption method, the ciphertext is multiplied by a large random number before it reaches the exponentiation method, so it appears that an adversary cannot interact directly with the secret values P or Q. The MontgomeryTimeser.exponentiate method does take a different path when a bit is set in one of the input arguments, but it looks like the adversary cannot control which path is taken."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"The application implements RSA using an algorithm (Montgomery multiplication) known to be vulnerable to timing side channels (MontgomeryTimeser). In particular, montgomeryMultiply leaks bits of a secret-derived value (this.M). The encrypted channel setup protocol also provides the attacker many opportunities to obtain timing observations due to the use of test encryptions as a mechanism for checking whether DH key exchange was successful in mutually deriving a shared session key. This protocol feature essentially serves as an encryption oracle that could be used as part of an attack to recover this.M and from there the secret required to obtain future plaintexts as a passive network adversary. Unfortunately, we were unable to obtain reliable timings within the operational budget when implementing this attack. Thus, we conclude that despite the theoretical presence of the vulnerability it does not in fact reach the required 80% success threshold."

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.3 BraidIt 4

A.6.4.2.3.1 Question 032 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"We observed that the size of the packets sent by the opponent changes based on the braid they select in the first round. By playing multiple games where an opponent selects a different braid in round one, we observe the size of specific packets sent from the opponent change which suggests the presence of a side channel in space. Each game between an attacker and an opponent generates 7 responses from the opponent to the attacker. As the opponent selects a different braid in round 1, we observe the 5th response packet from the opponent change in size. This should provide enough information to the attacker to query the oracle to find the opponent's chosen braids, b1 and b3, within the stated budget."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability as the braids are shuffled when presented to the attacker and the shuffling occurs in the target user's application.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"[…] While the group consensus for most of the engagement was that the attack could not be performed within budget, GrammaTech discovered that existing tools like javacg for recovering the state of a java.util.RandomRNG could be adapted to predict over a sequence of non-consecutive observations. In this case, javacg was successfully applied to the sequenced output ofnextInt(bound)while discarding the random bit output by nextBoolean()that is used to select upper- or lower-case letters in the generated braids. Therefore, an attack against the artifact can be successfully carried out by searching for RNG output sequences that match the received braids. For each individual string, a candidate state is found and then a prediction of the next sequence of random outputs is generated. This prediction is then matched against the other braid strings. If one matches, then this establishes a relative ordering between the first and second strings. By finding this ordering for each string, the attacker can derandomize the shuffled array indices, and use the space-leaked index to identify the modified braid. A PoC is available for this vulnerability."

Live-Collaborative Evaluation:

We believe that GrammaTech's exploit to undo the shuffle procedure is feasible within the operational budget. However, as the java random library is out-of-scope, we have noted this as an out-of-scope vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.4 Calculator 1

A.6.4.2.4.1 Question 014 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Vulnerable method: GreatNumber.divide. During execution, the divide method allocates many new arrays for storing coefficients derived from the dividend (~2000 in each iteration of the main loop; see GreatNumber.append and the GreatNumber constructor). Since the dividend is part of

the user input, this allows the user to influence the size of the allocated arrays and enables to force the JVM to use a lot of memory. If invoked with the dividend 10^499 and divisor 1, roughly 10MB heap memory is allocated per invocation. Hence, in the worst case, all available memory can already be consumed after ~400 divisions. As a results, the calculation of the expression '10^499'+'/1'*400 forces the JVM to allocate more than 4GB RAM on a new calculator instance."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The potential unintended vulnerability reported by Northeastern was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern was able to correctly identify an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.4.2 Question 016 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The transformMultiply method is used for multiplications of factors that have a degree greater than 1000 (i.e., numbers with more than 1000 digits). The way the algorithm tries to speed up the multiplication is by first transforming the input into the domain of complex numbers, then perform the multiplication with complex numbers and finally transform the result back. However, the transformation implemented in Transform.transform is convoluted and does not seem to improve the run-time compared to the naive easyLogspaceMultiply algorithm. A single multiplication of factors with a degree of 5000 already takes about a tenth of a second. Thus, to reach a run-time of 120s, the user merely needs to trigger a thousand multiplications with numbers in that order. The calculation of the expression '10^5000*10^5000+'*600 has a runtime of more than 120s on the reference environment."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The potential unintended vulnerability reported by Northeastern was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Northeastern was able to correctly identify an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.5 Calculator 2

A.6.4.2.5.1 Question 003 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"We could not find any indication for an algorithmic complexity vulnerability in space that would cause the challenge program to store a file with a logical size that exceeds the resource usage limit given the input budget. It is possible that the calculator creates a file based on user input:

- User input is fetched from the HTTP exchange with the CompositeAssistant class.
- This class relies on the package org.apache.commons.fileupload, which stores content either in memory, for smaller items, or in a temporary file on disk, for larger items.
- The threshold for storing in a file is set to 10KB.

However, we could not find a way to trigger amplification of the stored content size."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.5.2 Question 005 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Vulnerable method: BigInteger.plainLogspaceMultiply. The attack from Question_016 works completely analogous with calculator_2. This version of the calculator, however, also introduces an even more severe vulnerability in BigInteger.plainLogspaceMultiply, which is used for factors with a degree of less than 1000 (in contrast to transformMultiply). In the first version of the calculator, this method already has a quadratic run-time, but since the degree of factors is limited, it is not straight-forward to exploit. The same function in calculator_2, though, needs up to 7 times more time (based on our experiments). This is because, the main loop is artificially delayed and needs on average, four times more iterations including additional time for the Math.random calls. The calculation of expression '10^100010^1000+' 350 has a runtime of more than 120s on the reference environment."

Take-Home Evaluation:

The challenge contains an intended vulnerability. Northeastern identified an unintended vulnerability which was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program has an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.5.3 Question 009 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"We found that the same vulnerable function described for Question_014 can be used to exceed the resource usage of this question."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The potential unintended vulnerability reported by Northeastern was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program has an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.6 CaseDB

A.6.4.2.6.1 Question 019 (SC Space, Intended Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"We looked at all possible sources of network traffic and determined that there are side-channels in space that allow you to tell how many informants were returned as well as being able to reasonably guess the length of their names. However, because the length of informants in a case can be up to 10 characters, including but not limited to at least [a-zA-Z], that alone results in 56^10 = $3.033054891 \times 10^{17}$ different possibilities for only length 10 users. So, without a list of potential usernames that could be included, it seems impossible to obtain their identities."

Take-Home Evaluation:

The challenge contains an intended vulnerability that combines leaked information from the target user's request with a caching behavior and a deterministic algorithm to fetch documents to leak the secret.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern and other teams collaborated for a long time on various parts of CaseDB, but focused their efforts on Question 36, working minimally if at all on 19, the actual vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.6.4.2.6.2 Question 021 (SC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"The query engine called by CaseDBDocControllerHelper.runDocQuery when a user retrieves a case caches the query and its result. An attacker could leverage the caching to identify the queries performed by the target user. Specifically, the attacker can first enumerate all the cases (by searching for casename LIKE %) and then iteratively retrieve such cases and measure the response time for each query. By observing the fastest queries, the attacker can infer those most likely executed by the target user. Then, she can obtain a list of public informant names from the cases returned by the queries and ask the oracle for verification."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. Without further details on the analysis performed we do not believe the challenge is vulnerable to this potential vulnerability. Additionally, the description states that informants are secret i.e. there are no publicly available informant names therefore it is unclear how the attacker determines the list of secret informants.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"The secret can be leaked into the plan because the query selecting the CASE1X is cached from when the privileged user made it. However, it is filtered out before being sent. Queries such as using "like%" or>were not effective in influencing time consumption because these have not been run/cached with the secret in them."

Live-Collaborative Evaluation:

Through collaboration, Northeastern was able to correctly conclude that their potential vulnerability was invalid.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.6.3 Question 036 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"In order to view secret documents, an attacker needs to perform a request through the GET command, which requires privileged authentication. The GetCase.send method presents a side channel in time that would allow an attacker to leak the password one character at the time. However, the password in encrypted with the user secret key (and decrypted before the comparison). This does not allow the attacker to have control of the loop in GetCase.send and so makes the attack fail."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"In order to view secret documents, an attacker needs to perform a request through the GET command, which requires privileged authentication. The GetCase.send method presents a side channel in time that would allow an attacker to leak the password one character at the time. However, the password in encrypted with the user secret key (and decrypted before the

comparison). This does not allow the attacker to have control of the loop in GetCase.send and so makes the attack fail. We did find that the password is checked by decrypting the user's password with their key and a chosen IV. Then each byte is checked individually and an exception thrown for each one that doesn't match. If we could use timing to understand how many bytes were wrong then we could keep changing the IV until we reach the correct 8 byte password. However, we found no such timing difference. Even though the code is highly suspicious and looks vulnerable the time for an exception to be thrown seems too small to see in a real execution."

Live-Collaborative Evaluation:

Northeastern collaborated on this question for ~10 hours over the course of the live engagement and was able to correctly conclude the answer to this question, though with low confidence.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.7 Chessmaster

A.6.4.2.7.1 Question 006 (SC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"The time complexity is introduced by the artificial intelligence who plays against the player in a game. This AI always pick the best move given a specific board configuration and the time necessary for this decision greatly varies from configuration to configuration. It is possible to exploit this vulnerability in the following way:

1. Collect using passive operations the time for each move that the target player performs from the initial state to the state we are supposed to guess.
2. From the initial state get all the possible moves applicable to it.
3. Start a new game for each move retrieved in the previous point and collect the time response time for each one of them.
4. Discard all the responses with a response time which differs significantly to the one recorded at point 1.
5. Get all the possible moves from the remaining set of states.
6. Repeat from point 3 until as many times as the number of moves played by the target player.
7. Ask the oracle which one of the remaining states is the correct one."

Take-Home Evaluation:

We don't believe that there is sufficient separations in the timings for moves for the attacker to distinguish the next move with knowledge of the prior state and the response time. Without data to counter this argument we don't believe that the side channel identified is sufficiently strong.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"The application contains a side channel vulnerability in time that is introduced by the artificial intelligence who plays against the player in a game. This AI always picks the best move given a specific board configuration and the time necessary for this decision greatly varies from configuration to configuration.

Under the assumption that a game proceeds for at least 10,000 moves (as discussed in the live engagement), we could not identify a way to exploit this vulnerability to determine the game's state. For a normal chess game with fewer moves it is, however, possible to exploit this vulnerability in the following way:–Collect using passive operations the time for each move that the target player performs from the initial state to the state we are supposed to guess–From the initial state get all the possible moves applicable to it–Start a new game for each move retrieved in the previous point and collect the time response time for each one of them–Discard all the responses with a response time which differs significantly to the one recorded at point 1–Get all the possible moves from the remaining set of states–Repeat from point 3 until as many times as the number of moves played by the target player–Ask the oracle which one of the remaining states is the correct one"

Live-Collaborative Evaluation:

Northeastern correctly concluded that the game space clarification issued at the live engagement prevents a vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.7.2 Question 015 (AC Time, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"The time complexity attack is present in the autocorrect method and it can be triggered with large enough input size. The input budget specified in the question (20 kB) would be more than sufficient to exploit the vulnerability but a check that limits the input size to 30 characters avoids this problem. The highest possible number of iterations with that check is about 2629260, which is not enough to exceed the budget of the question. Moreover, it is possible to alter the execution time of the new game request adding more users to the system but the limit of 20kB indicated in the question is too strict to alter the runtime in any meaningful way."

Take-Home Evaluation:

The challenge contains an intended vulnerability through pawn promotion that allows an attacker to exceed the designed bounds of the AI's next move computation.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Due to an logical error, the application allows the user to promote a pawn to another king. With two kings on the board, the AI then takes significantly longer to compute a new move (1000+ seconds), because of the king's high piece score. The server can only handle one request at a time, thus also delaying any "new game" request for the required time. The logical error is introduced because the server first checks the user input if the name of the promoted piece is "king" and then auto-corrects it. This allows us to simply misspell the king (e.g., as "kin") which is then corrected to "king". A PoC is available for this vulnerability."

Live-Collaborative Evaluation:

Northeastern was able to correctly identify the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.8 Class Scheduler

A.6.4.2.8.1 Question 012 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"While the execution time depends on the number of students and teachers, it also depends on the number of rooms. Since the number of rooms is unbounded, i.e., the uploaded school file can contain any number of rooms, it is not possible to precisely distinguish the number of students and teachers."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"While the execution time depends on the number of students and teachers, it also depends on the number of rooms. Since the number of rooms is unbounded, i.e., the uploaded school file can contain any number of rooms, it is not possible to precisely distinguish the number of students and teachers. In the live engagement, significant effort was expended to explore this question. While one can observe how long each step takes, some part of the progress must be the number of teachers and number of students. With the help of Iowa, trials were run with multiple values of teachers, and we were able to predict generation 85% of the time. But, no consistent differences between different numbers of teachers and students, total time, slope of the line progress, were observed as all was consistent with some variance."

Live-Collaborative Evaluation:

Through collaboration, Northeastern was able to more confidently answer No correctly.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.8.2 Question 017 (AC Time, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"We could not find any algorithmic complexity in time that would exceed the resource usage of the download request. The method to download the schedule simply calls Schedule.toString, which walks through the CourseClass map and the number of courses/sections is limited during the upload."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"We found an attack in the genetic algorithm. The number of courses is bounded, but one can add a large number of sections to exploit a nested loop over the number of sections (see Schedule.generateFromChromosome, expensive loop in number of sections). By adding one course with -1 million sections and then one with 1 million sections, an AC/time attack was successfully produced. Because the server is non-blocking, the attacker must make multiple concurrent requests in order to successfully carry out the attack."

Live-Collaborative Evaluation:

The challenge does not contain an intended vulnerability. The reported unintended vulnerability was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.8.3 Question 028 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Assuming that the attacker does not have authorized credentials, we did not find any side channel in time using the available operations."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.9 Effects Hero

A.6.4.2.9.1 Question 010 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Although we can estimate the size of the processed file by observing the download response time, we believe there is no side channel in time which reliably allows an attacker to determine the content of the processed file. Also, guessing or leaking the URL of the downloaded file is not helpful since the server performs checks on the session ID."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.9.2 Question 025 (AC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"We could not find any algorithmic complexity in time since the user input does not affect the control flow of the program's initialization phase."

Take-Home Evaluation:

The challenge program did not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“The application is vulnerable to an algorithmic time complexity attack, which allows keeping the server busy for along time – e.g., using aBeatBlock with a high BPM setting and a long sample duration. However, since the server seems is non-blocking, these attacks cannot be used to delay the initialization of a different user.”

Live-Collaborative Evaluation:

Northeastern was able to elaborate on their reasons for saying No and answer correctly with more confidence.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.10 Railyard

A.6.4.2.10.1 Question 013 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

“We checked the loops identified by our static analysis tool and the portions of code that perform any kind of allocation such as Platform.AddCar and Platform.linkCars, and we could not find any amplification of the user input that can make the program exceed the resource usage.”

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“It is possible to extend the attack from Question 18 such that the application uses more than 5 GiB of memory. One key observation is that thePlatform.printCars method first prints the departing train to the log file (as leveraged in the attack from q18) and then prints everything a second time to a string buffer. Since the string buffer is stored on the heap, the attack would immediately follow using this second print, but because of the infinite loop, it is never reached. For the new attack, we thus aim to “break” the printing loop first and then create a new cycled cars list such that the program is stuck in another infinite loop that is writing to the string buffer. Therefore, the second train we use starts and ends with a coal car and includes several “padding cars” in between, e.g.,<□=J _] [O O] [O O] ... [O O] _]. After linking the first coal car, the printing loop continues with the cars in this new train, and if the printing loop can proceed faster than the linking loop, the next field of a car is null, and we successfully leave the loop. The control flow then proceeds to the second printToStream invocation and starts printing to the string buffer. In case the linking loop reaches the second coal car faster than the new printing loop, the attack succeeds and the application continues to print in an infinite loop to the string buffer. This causes the JVM to allocate more than 5 GiB memory. Our experiments show that this attack works reliably on the NUC, with 15 padding cars. A PoC is available for this vulnerability.”

Live-Collaborative Evaluation:

The unintended vulnerability confirmed by several teams was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.10.2 Question 018 (AC Space, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Similar to the previous question, we checked the loops identified by our static analysis tool and the portions of code that store content into files, and we could not find any amplification of the user input that can make the program exceed the resource usage."

Take-Home Evaluation:

Northeastern missed the intended vulnerability through the use of a static field to create a non-terminating loop.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"The application is vulnerable against an algorithmic complexity vulnerability in space. A programming error in the Coal_Car class allows creating a circle in the linked car list such that the program is trapped in an infinite loop during the "send out" phase of the train. In this loop, the program keeps appending to the internal log file, which eventually reaches a size of 5 GiB. Within the Coal_Car class, the flaw is introduced by mistakenly declaring the next field as STATIC. Therefore, all instances will share the same next field and consequently all coal cars in the train will point to the same car, which will cause the cycle in the linked list. Due to a quirk in the application, however, it is not possible to just add more than one coal car. This is because the application only uses the car type name for checking if a coal or passenger car is already in the train. Thus, this limit the numbers of cars with these types to one, but since there are two valid type names for coal cars (namely "coal car" and "coal_car"), it is possible to add two coal cars (one for each type) and mount the attack A PoC is available for this vulnerability."

Live-Collaborative Evaluation:

Following collaboration Northeastern identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.10.3 Question 035 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"There are 6 types of materials and, since each material can be either used or not, we have a total of $2^6 \equiv 64$ possible combinations. We found two side channels in space. First, in the Platform.AddCar method a programming error allows to add only one coal car and one caboose to the train. Hence, if the user tries to add additional cars of these types, we can detect this based on the sizes of the request and response, and by doing so we can conclude that passengers or coal could be in the train. Second, the response of the /api/v1/platforms/:name/cargo/add.json endpoint is much more verbose than to any other endpoint, allowing us distinguish this request based on its size and then reveal the number of cars in the train. However, these two side channels are not enough to reliably discover all the cargo materials without making further assumptions."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.11 SimpleVote 3

A.6.4.2.11.1 Question 001 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"The only possible attack vector is to obtain the registration key based on side channel leaks. While there is leak coming from the side channel of time from the LogicMapping class, for numbers that are less than 995, it is not possible to differentiate those numbers from each other because of the cache. Hence, we deem this attack to be impossible."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.11.2 Question 033 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"We analyzed the login handler (stac.webcontroller.manager.LoginManager.handleParcel, triggered by sending POST requests to /login). Our analysis shows that all loops are concretely bounded. There is no control flow branch that relies on the stored password. Furthermore, we measured the execution, and there is no measurable difference in time."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.11.3 Question 034 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"msgCreator.setSummary(ByteString.copyFrom((byte[])LZW.squeeze(structure, sumStr))); The "structure" is controllable by user. If we set it to the name of a candidate, the candidate's name will be compressed, and the size of the transferred data will be smaller, which is observable when two servers are communicating."

Take-Home Evaluation:

The challenge does contain a vulnerability, but it only leaks whether a candidate has received 100% of the votes.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"msgCreator.setSummary(ByteString.copyFrom((byte[])LZW.squeeze(structure, sumStr))); The "structure" is controllable by user. If we set it to the name of a candidate, the candidate's name will be compressed, and the size of the transferred data will be smaller, which is observable when two servers are communicating. However, with the help of other teams (Colorado, GrammaTech, Iowa, and Vanderbilt), we realized that the attack cannot work because the zipped "structure" is built differently. That is, the compression ratio is the result of an integer division, so it can only be either 0 or 1. This fact totally eliminates any possibility to have a candidate's name compressed together with the user's provided input."

Live-Collaborative Evaluation:

Through collaboration, Northeastern correctly concluded the challenge program was not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.11.4 Question 037 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"An attack closer to the CRIME attack exists in the stac.webcontroller.manager.TokenedPasswordChecker class. The added randomness is not sufficient and can be eliminated by retrying the login a few times and measure the length of the so-called "session string" (which was appended to the error path of the login failure page). Based on our experiments, 1500 queries are enough to get passwords of up to 100 characters."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The claim that the "randomness" is insufficient to prevent a vulnerability was further investigated. A single trial per character per position in a length-15 password (worst case secret) will use up $15 * 94 = 1410$ operations, leaving only 90 operations for additional trials, which isn't sufficient

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Vanderbilt and Iowa helped us on this question. In this challenge, the CRIME attack exists, but 1,500 queries are definitely not enough due to the existence of randomness (which we correctly identified in the original response). Un-fortunately, our original calculation was flawed: It requires approximately 100 * 2 attempts to recover one character of the password. As a result, 1,500 queries are not sufficient to recover passwords with a 90% success rate."

Live-Collaborative Evaluation:

Through collaboration, Northeastern correctly concluded that the challenge program was not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.11.5 Question 038 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The LogicMapping class implemented a hashing algorithm that is extremely slow for large numbers (especially if the numbers go beyond the ones stored in the cache). The isInBounds method in stac.easydecision.RegistrationGate class is the attacking target. We can build incorrect registration keys with its "core" numbers starting from 995 and onwards, pump up the MAX_N number to 1001, and then attack it by doing 1001 again. The limit on PDU is extremely tight in this case. We believe with some careful massaging of the HTTP header in our POST requests (i.e., removing unnecessary fields), it is possible to perform this attack."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. Attempts to craft exploits for the vulnerability reported by Northeastern required an input budget of 7652 bytes, which exceeds the input budget of 4 kB. During the collaborative engagement, Northern generated an exploit that satisfies the input budget.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"TheLogicMapping class implemented a hashing algorithm that is extremely slow for large numbers (especially if the numbers go beyond the ones stored in the cache). The isInBounds method in stac.easydecision.RegistrationGate class is the attacking target. We can build incorrect registration keys with its "core" numbers starting from 995 and onwards, pump up the MAX_N number to 1001, and then attack it by doing 1001 again. The limit on PDU is extremely tight in this case. We believe with some careful massaging of the HTTP header in our POST requests(i.e., removing unnecessary fields), it is possible to perform this attack. A PoC is available for this vulnerability."

Live-Collaborative Evaluation:

No change.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.12 SimpleVote 4

A.6.4.2.12.1 Question 023 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"The only possible attack vector is to obtain the registration key based on side channel leaks. While there is leak coming from the side channel of time from the ChaoticOperation class, for numbers that are less than 995, it is not possible to differentiate those numbers from each other because of the cache. Hence, we deem this attack to be impossible."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The potential unintended vulnerability reported by Northeastern will be investigated further.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that there was no vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.12.2 Question 030 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Similar to Question_038. The ChaoticOperation class implemented a hashing algorithm that is extremely slow for large numbers (especially if the numbers go beyond the ones stored in the cache). We can build incorrect registration keys with its "core" numbers starting from 995 and onwards, pump up the MAX_N number to 1001, and then attack it by doing 1001 again."

Take-Home Evaluation:

The challenge contains an intended vulnerability in triggering the application to clear the cache used to process registration keys, Northeastern's response is believed to be in line with the intended vulnerability although missing some details to fully confirm.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that there was a vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.12.3 Question 039 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“This question is similar to Question_034. However, its “structure” variable is constructed differently in this case, so it is not possible to reveal the names of the candidates.”

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that there was no vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.12.4 Question 041 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“The attacking target is the POST login handler. All reachable loops are concretely bounded, and our analysis does not find any reachable program location that may yield two distinct paths with a significant difference of execution time. Therefore, we deem this attack to be impossible.”

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.13 STAC Coin

A.6.4.2.13.1 Question 007 (AC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“The balance is not calculated every time like in a normal blockchain (looping through all the transaction of the wallet) but is calculated after every transaction. When the balance is retrieved the right amount expressed in integer is directly accessed through a HashMap. As a consequence, there is no algorithmic complexity in time.”

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“The balance is not calculated every time like in a normal blockchain (looping through all the transaction of the wallet) but is calculated after every transaction. When the balance is retrieved the right amount expressed in integer is directly accessed through a HashMap. As a consequence, there is no algorithmic complexity in time.

Iowa argued that the attacker (or a benign user) can spawn enough threads to use up the thread pool inside spark. The malicious client can create that many threads (200), and then not respond in any of the threads for a while to keep the threads alive, and hang the server. Unfortunately, we are not sure this is an “algorithmic complexity attack” (it is more likely to be a resource exhaustion attack), so we do not think this is in scope.”

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.13.2 Question 008 (AC Space, Intended Not Vulnerable, Take-Home: Yes, Live: No)*

Take-Home Response:

“A vulnerability is present because every time the blockchain is extended with a new block where this block extends the longest chain, the wallet of every node, containing the entire blockchain, is dumped into a file. We can make the blockchain grow to 1GB of size and then craft a special transaction, a transaction with the Genesis block as input, and announce it to the network. This will extend the blockchain and the target user will dump the wallet containing the entire blockchain (>1GB) to a file.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The potential unintended vulnerability reported by Northeastern is believed to exceed the input budget restriction provided by the challenge question.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

The unintended vulnerability is still not believed to fit within the input budget. Northeastern’s fellow collaborated noted in their response that the motivation was due to an interpretation of the wording of the input budget as request bytes rather than total bytes sent from attacker to victim. As this mis-interpretation motivated the “Yes” response of the teams’ collaborators, we believe that the same ruling should be applied to Northeastern. On the team’s take-home response, the motivation for the Yes response has not been noted as dependent on an interpretation of the question.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.13.3 Question 042 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“In order to spend coins of another user we need either the password or it’s private key. The private key seems impossible to leak or craft (it is generated using a standard algorithm) while the password is hashed using a custom hash. We carefully analyzed the implementation of the custom hash and we found it to have an issue with the constants/IVs (they are signed instead of being unsigned). However, this problem does not cause any side channel neither in time nor space.”

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable. Northeastern’s comments on the signed IV are in line with the comment may by GrammaTech concerning a potential vulnerability. Further investigation is required on this.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program was not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.14 Swappigans

A.6.4.2.14.1 Question 002 (AC Time, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

“If the username exists, the server computes the MD5 of the password. We can send a large password (up to ~35KB) to slow down the execution time. However, the budget is not enough to exceed the resource limit (6s).”

Take-Home Evaluation:

The challenge program contains an intended vulnerability in computing item purchase. The intended vulnerability enables an attacker to list an item that exceeds the intended maximum purchase price, allowing the attacker to exceed the resource usage limit as a result when attempting to purchase this item.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“An AC/time vulnerability exists in the calc method. A malicious user can create an account, add an item with the max price, create second account, add many items with low cost, and attempt to purchase the expensive item, thus triggering the vulnerability.”

Live-Collaborative Evaluation:

Northeastern’s response after collaborating does describe an unintended vulnerability identified by other teams.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.14.2 Question 020 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Knowing the username, we need the session key to make purchases. To get the session key we need the encryption key, which is generated randomly the first time the class SessionTokenManager is accessed. However, the seed uses a timestamp with granularity of seconds (System.currentTimeMillis() / 1000L). Hence, we can passively observe the login request at time t, generate (by applying the same algorithm of getSaltString) a set of possible keys with seed close to t (e.g., from $t - 500$ to $t + 500$), derive the session keys (by encrypting the target username) and ask the oracle for the correct one."

Take-Home Evaluation:

Northeastern flagged the same unintended out-of-scope vulnerability as several other teams. A key assumption of this vulnerability is the attacker knowing when the server was started. In future challenges we will work to clarify that the attacker does not know when the server was started.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Northeastern's unintended vulnerability was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.4.2.14.3 Question 031 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"While there is a timing side channel in the SwappigansItemMatcher.calc method, because the execution time depends on the price of the item to be purchased, the computation required to select the items to be exchanged strongly depends also on items that the target user has. Since this information is not known, an attacker cannot reliably identify the purchased item."

Take-Home Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Northeastern correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.5 University of Utah

A.6.5.1 Utah Overview

Utah answered 36 questions in the take-home engagement, 29 more than in the Engagement 5 take-home; however, this increase in number of questions answered was coupled with a 30 percentage-point decrease in accuracy (86% to 56%). The team had the lowest accuracy of all performers in the take-home. Utah had the lowest TPR in the segmentation of answered questions (13%), and the fourth highest TNR of the R&D Teams, 86%. In the segmentation of all questions (unanswered questions assessed as incorrect), Utah's TPR remained 13%, but the team's TNR decreased to 67%. The team had the most difficulty with AC questions achieving a 25% accuracy in AC Space questions and a 20% accuracy in AC Time questions.

During the live-collaborative engagement, all R&D Teams achieved between a 97% and a 100% accuracy. Utah achieved a 97% accuracy. Collectively, the R&D Teams identified unintended vulnerabilities in Railyard Question 013 and Class Scheduler Question 017 during the live collaborative engagement.

A.6.5.1.1 Take-Home Engagement Scores

Table A-133: Engagement 6 Utah Take Home Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	5	5	100
SC Time	9	7	78
SC Space/ Time	4	4	100
AC in Space	8	2	25
AC in Time	10	2	20
Total	**36**	**20**	**56**

Table A-134: Engagement 6 Utah Take Home Engagement Vulnerability Accurayc

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	15	2	13
Not Vulnerable	21	18	86
Yes Answer	12	2	17
No Answer	24	18	75

A.6.5.1.2 Live-Collaborative Engagement Scores

Table A-135: Engagement 6 Utah Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	5	5	100
SC Time	9	9	100
SC Space/ Time	4	4	100
AC in Space	8	8	100
AC in Time	10	9	90
Total	**36**	**35**	**97**

Table A-136: Engagement 6 Utah Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	15	14	93
Not Vulnerable	21	21	100
Yes Answer	14	14	100
No Answer	22	21	95

A.6.5.2 Utah Specific Responses

A.6.5.2.1 BattleBoats 3

A.6.5.2.1.1 Question 022 (AC Space, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"[…] the attacker can send a string with the shot parameters and the player calculates where the shot will land on his own machine (mainly because he needs the cannon coordinates). This calculation may be a possible vector for the attack.

The parameters are sent as a string which is then parsed on the victims machine. Two of the parameters are parsed to BigDecimal numbers, but before that a check is made to verify the strings consist of only digits, `-', and `.' characters. This removes the possibility of putting a number like 5E-100000 because of the stringent attack budget. Given an input budget of 10 kB, it is thus impossible to have an input with more than 10,000 digits.

The calculations to calculate the shot trajectory are quite complicated and could potentially involve large BigDecimals. However, the MathContext limits the precision of these calculations to 10,000 digits.

The calculation also has a hashtable trajectory, in which both the key and the value are BigDecimals. But the number of iterations is limited to 400, after which a default value is returned. For each entry, the key and value each have at most 10,000 digits, which means that the total space usage could be several megabytes, but filling the budget of 1000 MB seems out of reach.

Further calculations after the shot trajectory do not appear to be exploitable either. In summary we believe there is no algorithmic complexity vulnerability in space."

Take-Home Evaluation:

Utah incorrectly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"In the take-home portion of the engagement, we had suspected a potential vector of attack in this application. Specifically, we felt that the "shot" command could potentially be vulnerable to attack. The parameters of the "shot" command are sent as a string which is parsed on the victim's machine. First, the string is checked to ensure it consists only of digits 0-9, `-', and `.'. Then, two of the parameters are parsed into BigDecimal numbers. Because of the character check, we cannot input a number like 5E-100000 (which would be useful due to the attack input budget). Therefore, we have to write out the decimals in long form. Given an input budget of 10 kB, it is thus impossible to have an input with more than ~10,000 digits. The calculations to compute the shot trajectory are quite complicated and could potentially involve large BigDecimals. However, the MathContext limits the precision of these calculations to 10,000 digits. That said, MathContext is not used consistently and there are some typecasts to Double and typecasts back to BigDecimal that are potentially dangerous. The calculation also has a hashtable trajectory, in which both the key and the value are BigDecimals. But the number of iterations is limited to 400, after which a default value is returned. For each entry, the key and value each have at most 10,000 digits, which means that the total space usage could be several megabytes, but we don't believe this is exploitable. In the take-home portion of the engagement, we felt that we could not exploit this space usage within the input budget. However, Colorado (and, independently, GrammaTech) found a series of inputs which could cause the memory usage to exceed the budget. […] It is unclear as yet whether the shots must be input exactly as they appear here. Iowa did some further analysis and determined that the key parameter is the initialVelocity and the others can be changed to less complex values. Despite some collaboration with other teams, nobody has been able to discover the specific calculation that makes this exploit work. It is clearly related to the fact that Big Decimals are immutable and thus doing many calculations creates a large amount of them. Ensuring the loop runs several times and that the calculated number of have many decimals forces the benign client to store a large amount of BigDecimals, which are not collected sufficiently fast by the garbage collector."

Live-Collaborative Evaluation:

Working with Colorado, GrammaTech and Iowa, Utah identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.1.2 Question 026 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"[…] Using our tool we can select the "Boats" field and analyze its taint graph. This tells us what influenced the boat list (the method defineBoatsAndCannon) and what it influences

- Method OceanScreen.tagHitSquare and

- Method OceanScreen.updateHitsAsSunk

Both methods are invoked when determining the result of a shot.

They affect the size of the packets sent as a response to a shot, but not in any way that reveals special information. Method tagHitSquare changes a pin from MISS to HIT, this changes the size of the response but mainly just tells you what you already know (that you hit a boat).

If you sink a ship then updateHisAsSunk changes the pin from HIT to the corresponding ship, but if you sink a ship then you already know this information as well.

In summary it doesn't appear possible for there to be a side channel attack in space to determine the position of the boats."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.1.3 Question 040 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"[…] Using our tool we can look at the taint graph of the boats field and see that only the shot action involves the boats.

But, the position of the boats is agnostic to shot calculations. Every shot first calculates where it will hit (independent of the boats) and then checked against every boat. The only difference between hitting a boat or missing is setting a pin to either MISS or HIT, which doesn't affect running time significantly, and only gives you information which you already have (if you hit a boat or not.)

In summary it doesn't appear possible for there to be a side channel in time to determine the coordinates of the boats."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.1.4 Question 043 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"[…] If there is a chance the shot would hit the board then the cannon position is used to determine if it actually hit the board. At this point there are two possible code paths, and this might be used to determine the position of the cannon.

If the shot didn't hit the board, then there are no more calculations and there is an early return (the fast path).

But, if it did hit the board then several small calculations occur (the slow path)

- Determine which square was hit

- Determine if any of the neighboring squares was also hit

- Determine if a ship was hit

- Encode a slightly different answer than the other case

We measured the time taken between the two code paths in the best possible case (hit 4 squares, all the ships, and a large board) and where not able to detect the timing difference accurately. Since the calculation time is mostly spent on the trajectory calculation, and this presents some variability even with the same shot, and the extra calculations don't affect the time enough we believe that accurately detecting if the cannon is not possible with this method. In summary it doesn't appear possible for there to be a side channel in time to determine the position of the cannon. As a side note there does seem to be a side channel in space, the packet sizes when hitting the board/not hitting the board are different."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.2 BraidIt 3

A.6.5.2.2.1 Question 004 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The algorithm for reducing the braid is recursive for this attack we concentrated on finding a braid that will fail to reduce and in turn the algorithm will not terminate and keep printing out statement in the log file. For reducing the braid we call the normalizeCompletely method in the Plait class. normalizeCompletely in turn calls the freeNormalize method. In freeNormalize the code basically eliminate a sequence which has one lower case and one upper case character next to one another. […]

Now if we have a string which is like __x____X_____ where X is the endPosition character and x is the startPosition character the code will enter the if block. A check will happen on character and its inverse if they are not part of the the string between startPosition+1 and endPosition then we move ahead in the code. At this time a check will happen as to if the character-1 isAlphabetic. If the character is alphabetic a check will happen on character-1 or its inverse is not part the string between startPosition+1 and endPosition. […]

The newSection generated will be inserted into the string again in place of the string currently between startPosition and endPosition. At this stage normalizeOnce will complete and log a output in the log file. The control will go to normalizeCompletly which will call freeNormalize again. At this stage if isReduced is false again normalizeOnce and freeNormalize will be called again."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. There is an unintended vulnerability discovered by other teams, but the response by Utah does not indicate the same unintended vulnerability nor does it provide a mechanism by which the resource usage limit can be exceeded by a complex code structure within the application.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"There is a vulnerability in the app which can be used to form an attack. We will be using the Plait class for the attack Plait class has the method normalizeCompletely which has a while loop which is controlled by the response of isReduced method. The other two methods used are freeNormalize and normlizeOnce. The freeNormalize method has an out statement which print the concatenate braid in the log files. The normalizeOnce method does the braid reduction based if certain conditions are met. We wanted braids that will skip the for loop in freeNormalize method and print the out statement in the log and also not enter the eliminateSection block in normalizeOnce.

1. The braids have to satisfy certain conditions so that the app accepts them, the numStrand should be between 8 to 27 and the characters in the braids should be alphabetic and the inverse of every character should be between (char)(122-numStrand+2).
2. Once this is done the other aspect about the Plait class the loss variable which control the ground variable in the isReduced method which ensures that we enter the while loop always if the resulting ground variable is ensured to be in the braids and lowercase of ground variable is less than any other character in the brainds we will ensure that the while loop never exits in the normalizeCompletely method.
3. We now have to ensure that don't have and lower character and upper character of the same alphabet next to each other as freeNormalize will trim those.
4. After this the normalizeOnce method should not enter the eliminateSection block the method that controls this is the takeFollowingSection method and we can exploilt this method by passing a specific format of braid which does not have the character and inverse of the character in the braids and even if they do they should not satisfy the other condition mentioned in the loop in the takeFollowingSection. Since this is a peer to peer app the attacker can modify his code base and send the braids that will satisfy all these conditions.

5. All this conditions will ensure that the out statement in freeNormalize method will execute in an infinite loop."

Live-Collaborative Evaluation:

Working with Iowa and GrammaTech, Utah reported the unintended vulnerability identified by Iowa during the take-home.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.2.2 Question 027 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"The only way the attacker gets to determine round 1 and round 3 braids modified by the opponents is by offering the game to the opponent. Offer_game will cost one operation. At this stage the opponent will accept, modify and send the modified braid to the attacker. The attacker would want to know which braids were given to the opponent so they would print the braid, this will cost one operation. Once the attacker sees all the five braids they can ask the Oracle if the one braid at a time if that braid is the braid that was modified by the opponent. This will take a maximum of 4 operations. At this stage to continue the game the attacker will have to send the answer using make_guess operation. The attacker will now get braid options to modify and send to the opponent. To use the minimum number of operations the attacker will choose not to modify the braid and just use select_braid and send_modified. At this stage the opponent will answer the question and will be give the braid options for round 3. The opponent will send the modified braid to the attacker. The attacker will have to use the print operation again to print all the braids. The attacker will then query the Oracle which will take maximum 4 operations and to send the answer the attacker will use the make_guess operation. This brings the total operations required by the attacker to determine which two braids are selected and modified by an opponent to 15 operations."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable. Note, while the response alone doesn't indicate the entirety of the analysis performed, this response doesn't indicate analysis of observables in either a passively persistent or triggerable side channel.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.2.3 Question 029 (SC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"The secret is the plaintext messages sent between two benign users. The third-party attacker can figure out the secret as follows. Each game session has 3 rounds and each round has 3 steps. Player 1 selects a number of strands and sends it to player 2. To guess the number of strands, the attacker issues a strand number that is outside of the strand range, which will report an error from

that he knows the strand range is [8,27]. Then he issues at most 27-8+1-1 = 19 oracle queries in order to obtain the number of strands. This requires 19+1 = 20 operations in total. Player 2 generates 5 random braids, selects one of them and modifies it. The attacker can select a braid number that is outside the braid number range, which will trigger an error message (saying "braid_num must be 1, 2, 3, 4, or 5") leaking the choice of player 2, and the guess made by player 1 when player 1 guesses one of the 5 generated braids."

Take-Home Evaluation:

The challenge question's notional Oracle accepts queries of a set of candidate plaintext messages and provides a binary response as to whether the provided set is the correct set sent between the two benign users. Additionally, the attacker does not have a way to interact with the game between the two benign users under question. The response indicates the team did not properly parse the question.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"This attack is similar to the powerbroker attack in Engagement 4. To decrypt plaintext messages, we will be required to break RSA encryption by using the Chinese remainder method. Team from Colorado have discovered that if the guess is close to secret and the mod of the two is zero the app will take less time. The class to look in this case is the MontgomeryTimeser. The method that has this vulnerability is montgomeryMultiply and the point of vulnerability is the following line.

```
if (u.compareTo(this.M) > 0) {
   u = u.mod(this.M);}
```

However even with this vulnerability the total operations required to attack the app is more than 60000 and we have a budget of 40000."

Live-Collaborative Evaluation:

Working with GrammaTech, Colorado, and Northeastern, Utah correctly concluded that the challenge is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.3 Calculator 1

A.6.5.2.3.1 Question 014 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Class: com.cyberpointllc.stac.netcontroller.handler.MultipartAssistant

Most of the methods in this class are never called from anywhere in the code (as determined by our analyzer, cross-referenced with searching through the decompiled code), so I did not spend time analyzing them. The primary non-trivial method which is used is MultipartAssistant.fetchMultipartValues(...). The primary potential vulnerability that stuck out to me here was the presence of the Apache Commons FileUpload class. FileUpload is responsible for handling files which are uploaded to the server. It can cause files to be stored in memory if it is configured to do so and if the files are small enough (determined by a configurable threshold setting). The FileUpload class as used in the fetchMultipartValues(...) method is not configured

to handle actually transmitting files which are designated for upload. I did not see any other particular vectors for attack.

Class: com.cyberpointllc.stac.cruncher.Utility

The Utility class handles parsing input. The algorithms are small and straightforward and do not appear to be exploitable.

Class: com.cyberpointllc.stac.cruncher.Cruncher

The Cruncher performs actual computations for the calculator. There are two primary methods, both named processEquation(...). The implementations are identical with the exception that one uses the Cruncher instance's own Formatter for formatting the result, and the other takes a new Formatter as input when called. The processEquation(...) functions appear to be correctly implemented. There are no classes in use which could lead to an exponential blow-up in memory utilization, and the algorithms do not seem to unnecessarily create large objects."

Take-Home Evaluation:

Colorado and Northeastern identified an unintended vulnerability within the challenge application.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"There is a vulnerability in the GreatNumber.divide() method. Specifically, if the arguments are legitimate (no early return), a loop is entered over the number of coefficients. Within this loop is another loop that performs the computation of the division. The outer loop calls the GreatNumber.append() method twice, and the inner loop calls the same method twice. GreatNumber.append() is the site of the vulnerability. Each GreatNumber contains an array which contains all the digits of the number's value. Whenever GreatNumber.append() is called, a new array of the original length is initialized and modified. This array initialization is significant enough to cause the program to exceed the given resource budget if enough divisions are done with a sufficiently large base number. For example, 10^499 divided by 1 sequentially 400 times will cause the resource limit to be exceeded (i.e. an input of "10^499/1/1/1/...")."

Live-Collaborative Evaluation:

Utah appears to have collaborated with Northeastern as they reported the same unintended vulnerability discovered by Northeastern during the take home.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.3.2 Question 016 (AC Time, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

The server used by the challenge program is single-threaded, which means the only way to delay processing of a benign request would be to hang the server's operation indefinitely (thus preventing subsequent requests from being handled at all, much less in the specified time allotment). Many of the endpoints in this application are trivial to analyze. Of the rest, most of them share calls to a few select "helper" classes. We analyzed most carefully the helper classes, and any algorithms encountered along the way. All algorithms used appear to be written correctly, such that they cannot be made to loop forever. (Consider, e.g., Karatsuba fast

multiplication in mathematic.OptimizedProductGenerator.) Therefore, these cannot be exploited to hang the server. Exceptions which are generated during things like computing results, parsing number values, etc., are all wrapped in try/catch blocks where appropriate, such that the server cannot be made to throw an exception and then hang. Therefore, we conclude that there is no vulnerability which would satisfy the question."

Take-Home Evaluation:

Colorado and Northeastern identified unintended vulnerabilities in the challenge program. Note, hanging the server indefinitely is not the only way to delay a benign response. Any request that takes more than the finite resource usage limit will suffice.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"The application does not process parentheses in the expressions in the expected manner. In traditional math, the expression "(1 + 2) (3 * 4)" should yield the result 36. However, calculator_1 will just give back 12: the result of the final parenthetical expression. This is despite the fact that the previous expressions are computed (i.e., their results are computed and then abandoned). This occurs because of the way Cruncher parses its procedures. To exploit this, we simply need to wrap an expensive operation in parentheses and duplicate it a large number of times. From work on other questions, we know that exponentiation and multiplication are somewhat slow. A particularly slow computation is "999^1695 * 999^1695 * 999^1695". This expression can be wrapped in parentheses and duplicated 350 times (i.e., "(999^1695*999^1695*999^1695) (999^1695*999^1695*999^1695) ...") without any intervening operations. Such a request causes the challenge program to take longer than 120 seconds on the NUC."

Live-Collaborative Evaluation:

Utah collaborated with Northeastern, Iowa, Colorado, and Vanderbilt to identify the unintended vulnerability flagged by some teams during the take-home.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.4 Calculator 2

A.6.5.2.4.1 Question 003 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Investigation of the external APIs used by the application reveals a few calls to functions which may allow file storage, namely:

- java.io.OutputStream.write()
- java.io.Writer.write()
- java.nio.file.Files.copy()
- org.apache.commons.io.IOUtils.copy()
- org.apache.commons.fileupload.*
- org.slf4j.Logger.*

The OutputStream is only used for writing to StringWriters, so it is of no concern. The same is true for Writer. Files.copy() is a bit more suspicious, since it appears in code in the CompositeAssistant class's pullCompositeFile() method which clearly handles file uploads and

would likely be used to create files on the server. However, the pullCompositeFile() method is never called anywhere; it appears to be a dummy method. Our static analysis confirms this suspicion. IOUtils.copy() is only used once in authorize.KeyExchangeVerifier() where its argument for output is an instance of StringWriter (not a file). The org.apache.commons.fileupload.* methods are somewhat more suspicious. These consist of various methods and classes whose express purpose is to facilitate a web server's ability to receive file uploads from a client. A closer look shows that either code is written which would handle file uploads but is never called (similar to Files.copy() above), or else the methods/classes are only used for things like parsing HTTP header information without actually initiating any receipt of an upload. Lastly are the org.slf4j.Logger.* calls. A log file is created on or shortly after startup. However, there are only two calls to write to the log, and both of them depend on the failure of a Java Future object to deliver on its promise. This behavior is not manipulable by the end-user."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"[…] The ExecutionException is raised when a Future's process raises an uncaught exception. The application successfully handles these exceptions -- there isn't a way to crash the server. However, the stack trace of uncaught exceptions gets outputted to the log file. There is a way to generate a StackOverflowException during computation of an expression if a sufficiently large number of parentheses are used. Specifically, sending an expression of 9000 left-parens, a valid internal expression, and 9000 right-parens will cause the call stack to overflow, thus leading to output being logged to file. However, in the best case, 76KB of data are written to disk per exception, at a cost of ~20KB of input data. Three exception-causing exceptions can be transmitted (staying within budget), but no more than ~230KB of data will be written to disk. We have not been able to identify any other method of causing uncaught exceptions to be generated during execution of the calculator, so we believe there is no exploit for this vulnerability within the given input budget."

Live-Collaborative Evaluation:

Utah worked with Northeastern and Iowa to confirm their non-vulnerable response.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.4.2 Question 005 (AC Time, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

The server used by the challenge program is single-threaded, which means the only way to delay processing of a benign request would be to hang the server's operation indefinitely (thus preventing subsequent requests from being handled at all, much less in the specified time allotment). Many of the endpoints in this application are trivial to analyze. Of the rest, most of them share calls to a few select "helper" classes. We analyzed most carefully the helper classes, and any algorithms encountered along the way. All algorithms used appear to be written correctly, such that they cannot be made to loop forever. (Consider, e.g., Karatsuba fast multiplication in algorithm.OptimizedTimeser.) Therefore, these cannot be exploited to hang the

server. Exceptions which are generated during things like computing results, parsing number values, etc., are all wrapped in try/catch blocks where appropriate, such that the server cannot be made to throw an exception and then hang. Therefore, we conclude that there is no vulnerability which would satisfy the question."

Take-Home Evaluation:

Utah incorrectly concluded that the challenge program is not vulnerable. Note, there exists a threshold for vulnerability between looping forever and not looping forever. The challenge contains an intended vulnerability in parsing input expressions.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"The BigInteger.multiply() method does a check to determine whether the degree of both arguments of a multiplication is greater than 1000. If at least one of the arguments has degree 1000 or less, BigInteger.plainLogspaceMultiply() is called. BigInteger.plainLogspaceMultiply() iterates over the sum of the degrees of the two arguments, and then has three nested while loops. These loops govern the iteration over the coefficients of each number, which are used to compute the product. The inner two of these while loops have a secondary condition: that Math.random() be less than 0.5. The effect of this is that the computation takes (on average, statistically) four times as many iterations as it would otherwise. To exploit this vulnerability, large numbers must be multiplied together. To save space on the input, we pass these large numbers indirectly by means of exponentiation. The example deduced to prove the exploit is: 10^1000 multiplied by itself 599 times. When given this input on a NUC, the application takes at least 120 seconds to complete the request."

Live-Collaborative Evaluation:

Utah worked with Northeastern, Iowa, Colorado, and Vanderbilt to identify one of the unintended vulnerabilities in the challenge program.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.4.3 Question 009 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Most of the endpoints in the application are handled with classes that contain trivial code. None of these provided any vulnerabilities. Some endpoints make calls to the webcontroller.coach.CompositeAssistant class's fetchCompositeDetails(...) method. This method uses the Apache Commons FileUpload class, which is used for handling file uploads from a client. The FileUpload class can be configured to store smaller files in memory until they are handled. This configuration has not been performed, so the challenge application's use of FileUpload is not vulnerable to exploitation. There are also calls into the mather.Utility class, which handles parsing inputs. The algorithms used are small and straightforward, and do not appear to be vulnerable in the desired way. The Mather performs actual computations for the calculator. There are two primary methods, both named calculateEquation(...). The implementations are identical with the exception that one uses the Mather instance's own Formatter for formatting the result, and the other takes a new Formatter as input when called. The calculateEquation(...) functions appear to be correctly implemented. There are no classes in

use which could lead to an exponential blow-up in memory utilization, and the algorithms do not seem to unnecessarily create large objects."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability; however Northeastern, Two Six Labs, and Iowa identified unintended vulnerabilities.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"There is a vulnerability in the BigInteger.divide() method. Specifically, if the arguments are legitimate (no early return), a loop is entered over the number of coefficients. Within this loop is another loop that performs the computation of the division. The outer loop calls the BigInteger.append() method twice, and the inner loop calls the same method twice. BigInteger.append() is the site of the vulnerability. Each BigInteger contains an array which contains all the digits of the number's value. Whenever BigInteger.append() is called, a new array of the original length is initialized and modified. This array initialization is significant enough to cause the program to exceed the given resource budget if enough divisions are done with a sufficiently large base number. For example, 10^499 divided by 1 sequentially 400 times will cause the resource limit to be exceeded (i.e. an input of "10^499/1/1/1/...")."

Live-Collaborative Evaluation:

Utah collaborated with Northeastern and Iowa to identify one of the unintended vulnerabilities in the application.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.5 Chessmaster

A.6.5.2.5.1 Question 006 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"First, we consider the passive operations. The question states that the user does not issue any moves while the attack is ongoing. Thus the only request that the attacker could passively observe is the "display" request. This calls "this.client.state.display()" which indeed depends on the board state. However, "display" is a simple function with two loops that both run a fixed number of iterations (i.e., 8 times). Thus its runtime does not depend on the contents of the board. Thus the passive operations do not leak any information about the board state.

Second, we consider the active operations. The board state is stored in the "state" field of "GameClient". The "GameClient" is stored in the "client" field of "User". The "User" is stored in the "usernames" field of "Server". Thus in order for an active operation to leak information, the attacker's action must depend in some way on these fields. There is indeed a "client" field in "Server" that saves the last "GameClient". However, this field is overwritten once a login succeeds. Thus exploiting this field must happen before that overwrite. However, the "client" field of "Server" is not referenced before that overwrite. Thus, information cannot be leaked this way.

There are no other paths by which the information could leak, so there is no such vulnerability."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable. A clarification was issued stating that rather than “The target user is in the process of playing a game and does not make a move during the attack”, instead “The target user is in the process of playing a game and will only make moves 2 minutes after the server's response to the target user's previous move.”

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“During the live portion of the attack, it was discussed whether one could discover the state by the amount of time that the AI player takes to make each move. The AI player does take a different amount of time depending on the current board state. The sequence of timings between moves does result in a fingerprint as to the sequence of moves. However, there are still states that have the same or similar fingerprints. This state space explodes very quickly as the number of moves progresses. After 10,000 moves, it is unlikely to be possible to still distinguish between states. A more definite answer would require experimentally measuring all possible opening moves by the human player and measuring how distinguishable the timings are. However, we do not have the processing power to do this. (Testing even just 4 moves from the human would involve billions of possible states.)”

Live-Collaborative Evaluation:

Utah performed additional analysis during the collaborative engagement to affirm their “No” response.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.5.2 Question 015 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“The server is single threaded. Thus, the benign request must wait until any requests submitted before it. To delay the "new game" request, the attacker merely need to submit a request that takes a long time to process before the benign request can be processed. Every word of request is sent through Autocorrecter.correct. This function constructs a list of possible edits by building strings that have a single character changed. Each possible string is of size n and there are n possible strings, thus building these strings takes quadratic time. Thus, to make a request take a long time, the attacker simply submits a request containing a long word such as a single word that is 19,000 characters long.”

Take-Home Evaluation:

The challenge program contains an intended vulnerability. Utah reported a potential unintended vulnerability in spell correction which is prevented by an input guard.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“First, we find a sequence of moves that allows us to promote a pawn. The AI is deterministic so the attacker can find such a sequence by simulating the application in the attacker's machine. Next, we promote the pawn to a "king". Normally this is prevented by a check whether `lines[4].toLowerCase().equals("king"))`. However, we can bypass this check by misspelling "king" as "kng". Immediately after that check, there is a call to the AutoCorrector on that string. This then corrects "kng" to "king" or "KING" depending on whether the player is white or black.

This then causes the pawn to be successfully promoted to a king. In State.totalPieceScore(), each king increments the score of that board by 20000. With multiple kings on the board, this causes the board to score much higher than expected. This in turn causes the AI in GameClient to search much deeper into the game tree. As a result, this takes an amount of time that exceeds the time budget for this question, and the attacker's request is delayed until this is completed. Since the server handles requests in a blocking fashion, the benign request will be blocked until the attacker's request is finished being served, and thus the application is vulnerable."

Live-Collaborative Evaluation:

Utah worked with Colorado and GrammaTech to identify the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.6 Class Scheduler

A.6.5.2.6.1 Question 012 (SC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"Without school administrator credentials, the attacker cannot access parts of the app gated behind `this.AUTH.isUserLoggedIn()` tests. Furthermore, this question does not have any passive operations, so any side channel would have to be in those parts of the app accessible without logging in. The only candidate for this is the callback code for generateScheduleButton due to it checking user authentication late and the code controlling the progress bar based on the progress that has been made in the last. The progress bar is slow and its speed is significantly influenced by the number of teachers and students. We are still working out the details, but we believe this could lead to an attack."

Take-Home Evaluation:

The challenge question allows passive operations. We do not believe there is a viable avenue of attack based on the progress bar. This is associated with the schedule generation time and we believe that there are too many variables that contribute to equivalent response times to allow for this avenue of attack.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"[…] However, after extensive statistical testing in collaboration with Iowa, it does not appear that there is a reliable or significant enough difference between a large or small number of students. There is some difference, but it is small and is obscured by other sources of noise in the system."

Live-Collaborative Evaluation:

Utah collaborated with Iowa to correctly determine that the multiple sources of the schedule generation time prevent the avenue of attack proposed in their take-home response.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.6.2 Question 017 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“A benign request can be delayed as follows. An attacker can successfully upload a malicious schoolData.xml (shown below) where there are 56,000 "lol"s (7*8*10*10*10) in the courseId element of a teacher's courses list, and then generate a schedule based on that data file. However, the process of generating such schedule will never terminate, which will block a benign request when the classscheduler server is single threaded.”

Take-Home Evaluation:

The challenge is provided as delivered. The attacker does not control the server start parameters. Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Normally, the maximum number of courses is limited by maxSlotsAvailable in VaadinUI.validateData(). This is the sum of the getNumSections() of each course. However, there is no check to ensure this value is positive for each course. So, if you add a course with a large negative number of sections and a course with a large enough positive number of sections such that the sum of the two is greater than zero and less than maxSlotsAvailable, they will cancel each other out, and the result will pass the check. This allows us to upload a class schedule with a course that has a large number of sections. Next, we use this to tie up the server by clicking the button to generate a schedule. The schedule generation process involves a significant amount of computation whenever a particular course has a large number of sections. Finally, we have to deal with the fact that the server keeps multiple threads for processing requests. Fortunately, in our tests, we found that sending multiple requests from multiple different sessions to generate a class schedule allowed us to lock up all the available server threads. At this point, the response to the benign request would get delayed while the server is waiting for processing threads to become available.”

Live-Collaborative Evaluation:

The challenge does not contain an intended vulnerability. The unintended vulnerability reported by several teams was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.6.3 Question 028 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“Without school administrator credentials, the attacker cannot access parts of the app gated behind `this.AUTH.isUserLoggedIn()` tests. Furthermore, this question does not have any passive operations, so any side channel would have to be in those parts of the app accessible without logging in. The only candidate for this is the callback code for generateScheduleButton due to it checking user authentication late. While this code checks the authentication late, none of it is influenced by a particular teacher's schedule. Thus, there is no attack.”

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable. However, the team’s response does not indicate exploration of any avenues of attack to leak the required credentials.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.7 Effects Hero

A.6.5.2.7.1 Question 010 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“This question asks if there is a side channel in time that allows an attacker to determine the entire contents of the file returned to a target user. One possible way is to determine the file uploaded and every block that is added and their configurations. We don't think this is possible, since most of the blocks are similar and there is no way of determining their configurations. The time taken to configure the blocks is heavily user dependent. A second possibility is to measure the time taken to process the target user's request. This also depends on their choice of blocks, and their configurations, but even if we ignore this issue we don't think there is an attack. For every block that is processed, there are no significant differences in the computation path that is taken, regardless of the actual values in the sample[] array. Since the output file is encrypted over the HTTPS connection, an attacker has no way to determine what these values are.”

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.7.2 Question 025 (AC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

“[…] We have observed the following characteristics of the program that can be exploited by an attacker.

The block that takes the longest to process is the BeatBlock, for two different reasons. For the appropriate choice of parameters (BPM = 330,750, duration = 1 millisecond), processing a BeatBlock can trigger around 44,000 lines of errors messages to be logged. It also triggers seven calls to fade(), which does various matrix operations, including finding the determinant of a 6x6 matrix using the method of expansion by cofactors. On a desktop machine, each call to fade() takes around 600 milliseconds.

We then must determine how to force the program to process BeatBlocks as many times as possible. The program only allows a total of 50 blocks to be added to the stage. We first add a

chain of 24 beat blocks, where beat1 takes beat2 as input, beat2 takes beat3 as input, etc. We then add single sine wave, and make it the input to beat24. We then add 25 more beat blocks, each of which has the output name N/A and input the sine wave.

The first 24 beat blocks and the sine wave cause the while loop in VaadinUI::runStage() to run 25 times. Each time, a single output name is added to the set of blocks that have been run, and keeps the loop running.

Also during each loop, because outputs labeled N/A are a special case, every one of the 25 BeatBlocks with the output name N/A will be processed.

Thus, BeatBlock::process will be called a total of 649 times for each second. By setting the slider to 30 seconds, this causes BeatBlock::process() to be called 19,470 times. Each of these calls fade() 7 times, for a total of 136,290 calls, taking around 81 seconds. In addition, each time outputs 44,000 lines of error messages, for a total of over 856 million lines of errors."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The unintended vulnerability reported by Utah was identified by several other teams, and we note that it results in high CPU consumption; however, as observed by Two Six Labs, this identified vulnerability is not believed to sufficiently impact the benign request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Our original attack described below works to tie up the server, but apparently Vaadin UI uses a thread pool that defends against the effect. When we tested it before it we used only a single browser where it seems the attack works, but this is an artifact of using Vaadin Ui which seems to behave differently if the attacker and client are on the same browser and if they are not. Further testing relieved that if they are not on the same browser than the attack doesn't work. Finally, we note another possible interesting attack found by Iowa. They used their tool Atlas to find that FileInputBlock has a suspicious function getNextFloat(). The main idea is to upload a small WAV file which has the maximum number of channels possible (around 32k). This function iterates over every channel several times. By correctly setting a few other parameters this function is vulnerable and takes well above the resource usage limit. Unfortunately, this runs into the same problem with our attack in that it only works if the attacker and client are on the same browser. […]"

Live-Collaborative Evaluation:

Utah worked with all other R&D Teams to confirm that the vulnerability reported during the take-home does not impact the benign request response.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.8 Railyard

A.6.5.2.8.1 Question 013 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The attack for this question is based on the attack mentioned in Question_018. Once we execute the attack for Question_018 the string stream will throw an exception and will remain in the heap

memory this is because the string stream is not included in the try block hence will not be closed at the end of the try block and will not be eligible for Garbage Collection."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The linear complexity with regard to user input without creating a looped construct in the parsing of input expressions is believed to render the application non-vulnerable to the potential vulnerabilities reported in the Question 018 and 013 responses. Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"For this attack we will use two platforms A and B for platform A we will add two coal cars and form a chain. When we send out the train the app will enter an infinite loop because of the chain caused by static next field in coal car class details of which are mentioned in Question 18. At this point we add a coal car followed by 100 cargo items followed by another coal car in Platform B this will break the chain for the infinite loop running for Platform A in the printToStream method using the file writer as the next field is now point to null and it will exit method printToStream for the file writer and Platform A will enter the printToStream for the string writer. Once this happens the string writer for Platform A will go into the loop to print the cargo being added and this will give enough time to link the second coal car being added in Platform B once that is added we will have a chain again and the print for string writer will go in infinite loop and we will use 5GB of heap memory."

Live-Collaborative Evaluation:

This approach to triggering an unintended vulnerability appears to have come out of a joint collaboration of all R&D Teams as no team identified this unintended vulnerability during the take-home engagement.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.8.2 Question 018 (AC Space, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"We will utilise two vulnerabilities in the application for this attack. The first vulnerability is the ability to pack more elements in a bin than the capacity of the bin. This is possible if we use the BEST_FIT packing method in the application. The algorithm iterates through the list of particular cargo to be packed eg. Coal cargo. The algorithm will try to find the best fit among all the coal bins given the cargo quantity and the remaining cargo capacity for a bin. After placing the current cargo the algorithm updates the remaining capacity of the bin by subtracting the current cargo quantity with the remaining quantity of a bin. However, no negative check is done and we can subtract a negative quantity from the remaining quantity of the cargo and the result is a huge remaining capacity for the bin which we can use in subsequent cargo addition and thus pack more elements than allowed in a bin. The second vulnerability is the algorithm to send out a train. The SendOut method in Platform class calls the printCars method by passing a file stream and a stringstream to print the cars and the resulting cargo. This call is surrounded by a try-catch-finally block. The printCars method uses the file stream to call the printToStream method this will output the given train into out.txt file. The second call by printCars to printToStream method

uses the string stream. Now during the call to printToStream using the string stream if we supply the string stream with $2^{31} - 1$ characters the index for the string stream will become negative and it will throw an IllegalArgumentException the code will break and go to printCars which will in turn go to SendOut method. In SendOut method the finally block will be executed and the cars attached to the train are wiped however the Cargo, Personnel and Schedule list is still present. Now we can attach just one add car request and one send out request and it will use the added Cargo, Personnel and Schedule from previous request to fulfill the current send out request. This way we can keep on appending to the out.txt file."

Take-Home Evaluation:

The challenge contains an intended vulnerability that allows for a cycle. The linear complexity with regard to user input without creating a looped construct in the parsing of input expressions is believed to render the application non-vulnerable to the potential vulnerabilities reported in the Question 018 and 013 responses.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"For this question we will use the vulnerability in Coal_Car where the next field is static and also the fact that we can add two coal cars in the same train by using car_type as coal_car and coal car. The way to exploit this vulnerability is once we add all these cars we will be linking these cars in the linkCars method of the Platform class. The linkCars method uses reflection to call the respective car classes we have already added coal_car and coal car as cars and as the next field is static for coal car even when we link the second coal car it will point to the next field of the first coal car thus forming a chain. After adding and linking the cars in this fashion we can then send out this train this will trigger the printing in the out.txt. The method used for printing is printToStream which has a do..while loop controlled by the current car the loop is iterating over however as we have formed a chain using the static next field this loop will go forever and generate 5GB out.txt required for the attack."

Live-Collaborative Evaluation:

Utah collaborated with Iowa, Vanderbilt, GrammaTech, Northeastern, and Colorado to identify the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.8.3 Question 035 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Active Operations: Only the railyard manager can view the cargo being loaded into a train. As the during the schedule is being generated no other manager authenticates herself viewing the cargo by issuing a request while the cargo is being loaded is not possible. The attacker is unprivileged so viewing the cargo by generating the schedule of recently scheduled train which displays the cargo loaded in the train is not possible. Oracle Queries: There are 6 types of cargo paper lumber, crates, coal, livestock, passengers. 64 combinations of cargo is possible which is less than the given 14 operations. Passive Operations: The attacker has no control on the activity of railyard manager and by observing the traffic will not be able to conclude the cargo that is being sent in 14 operations."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable. This response doesn't indicate any analysis of packet traffic in an attempt to rule out passively persistent side channels.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.9 SimpleVote 3

A.6.5.2.9.1 Question 001 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"After examining the code on the server that updates a user's ballot, there is a difference in execution path depending on whether the user has altered the ballot each time it is viewed. If it has been altered, then the database has to be updated for both the ballot and the voter.

After each ten updates to the database, the results for all currently active referendums are recounted and sent to the other server. The server also sleeps for 300 milliseconds after each of these recounts.

As mentioned in Question 34, the ElectionResultsMessages that are sent between servers are padded, so the size does not give away any information about the content. But the locally stored ReferendumResults are not padded. In particular, if someone votes for a choice that has not been voted for before, the time and space usage on the server will increase slightly over the previous update.

However, we believe that the difference in space will be only a few bytes, and the difference in time will be similarly minimal. This is not enough for the attacker to observe the change reliably.

Therefore, we conclude that an attacker cannot obtain a ballot belonging to another voter."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.9.2 Question 033 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“[…] 1) First we examined the LoginManager, to see if the attacker could attempt to determine the user's password directly. When he tries to log in with the target voter's username but with an incorrect password, a token is constructed by the TokenedPasswordChecker and returned as a URL argument.

This seemed suspicious, because it is not necessary to display this token to the user. However, after further investigation, we concluded that this token is essentially random. Before the token is displayed to the user, the function LoginManager::pullTokenByte() takes each digit in the token, adds a random number from 0 to 9, and then returns the result mod 10.

Therefore, unless the attacker could break the random number generator, he gets no information from this token except (possibly) the length of the password. The attacker could also attempt to use the time taken to respond to the login request. But once again, the time depends only on the length of the password, not the actual characters that it contains.

2) Next we looked at how the server responds to a request for a profile. This is handled by the VoterManager, which takes the customer ID for the session from a cookie and then queries the voter database with that ID.

So if the attacker could manually set his cookie to include the target voter's ID, then the server would return the target voter's profile. But this ID is generated as a random UUID during each session, and never affects the runtime on the server. Therefore, this approach is not feasible either.

3) Lastly, we looked at the voter database itself, to see all of the ways it can be accessed. But in each case, the time used by an access to the database will never depend on the value of the profile that is returned.

After trying each of these three avenues of attack, we conclude that there is no side channel in time by which an attacker can view a target voter's profile.”

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.9.3 Question 034 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“[…] First, he could try to log in as an administrator. But as discussed in our answers to Questions 33 and 37, we believe that the tokens returned on the login page cannot be used to log in as a different user, even if their username is known. Secondly, he could monitor the traffic between the two election servers, each of which stores partial results. To generate the full results, they exchange an ElectionResultsMessage. Each of these messages will always include 20 AnswerTally messages, each of which will have ID's for five possible answer, and counts for the

number of votes for each answer. If there are fewer than 20 questions on the ballot in an election, or fewer than five choices for a question, the fields will be set to default values. Each answer ID or vote count is also padded to use exactly five characters, regardless of the actual value. To conclude, every ElectionResultsMessage will have exactly the same size, regardless of which election is being processed or what the results were. Therefore, we do not believe that the attacker can obtain any private information about the results from analyzing the space usage."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"There are at least two ways in which an attacker could attempt to discover the results of the election. First, he could try to log in as an administrator. But as discussed in our answers to Questions 33 and 37, we believe that the tokens returned on the login page cannot be used to log in as a different user, even if their username is known. Secondly, he could monitor the traffic between the two election servers, each of which stores partial results. To generate the full results, they exchange an ElectionResultsMessage. Each of these messages will always include 20 AnswerTally messages, each of which will have ID's for five possible answer, and counts for the number of votes for each answer. If there are fewer than 20 questions on the ballot in an election, or fewer than five choices for a question, the fields will be set to default values. Each answer ID or vote count is also padded to use exactly five characters, regardless of the actual value. To conclude, every ElectionResultsMessage will have exactly the same size, regardless of which election is being processed or what the results were. Therefore, we do not believe that the attacker can obtain any private information about the results from analyzing the space usage. Finally, we note that the approach used in last engagement, and which other teams tried, doesn't work. The INCLUSION_THRESHOLD is always compared to an integer so there seems no way of using this approach."

Live-Collaborative Evaluation:

Utah collaborated with all other R&D Teams and explored the potential avenue of attack in the INCLUSION_THRESHOLD identified by several teams during the take-home.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.9.4 Question 037 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"1) First we examined the LoginManager, to see if the attacker could attempt to determine the user's password directly. When he tries to log in with the target voter's username but with an incorrect password, a token is constructed by the TokenedPasswordChecker and returned as a URL argument. This seemed suspicious, because it is not necessary to display this token to the user. However, after further investigation, we concluded that this token is essentially random. Before the token is displayed to the user, the function LoginManager::pullTokenByte() takes each digit in the token, adds a random number from 0 to 9, and then returns the result mod 10. Therefore, unless the attacker could break the random number generator, he gets no information from this token except (possibly) the length of the password. The attacker could also attempt to

monitor the space usage on the server to break the password. But once again, the space depends only on the length of the password, not the actual characters that it contains.

2) Next we looked at how the server responds to a request for a profile. This is handled by the VoterManager, which takes the customer ID for the session from a cookie and then queries the voter database with that ID. So, if the attacker could manually set his cookie to include the target voter's ID, then the server would return the target voter's profile. But this ID is generated as a random UUID during each session, and never affects the space usage on the server. Therefore, this approach is not feasible either.

3) Lastly, we looked at the voter database itself, to see all of the ways it can be accessed. But in each case, the space used by an access to the database will never depend on the value of the profile that is returned. After trying each of these three avenues of attack, we conclude that there is no side channel in space by which an attacker can view a target voter's profile."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.9.5 Question 038 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"With only 4 kB of input, an attacker needs to delay a request from a benign user to view a ballot for more than 300 seconds. We discovered that there is a trapdoor in checking the registration keys. Before a key is checked in the database, it is first processed by the RegistrationGate. This gate includes a trapdoor that can be triggered under a very specific set of parameters. By submitting the set of registration keys below, we caused the server to run for multiple hours, which is far longer than the required time of 300 seconds.

[…] The pattern of registration keys was chosen carefully to pass through several conditions in RegistrationGate.check. First, the array chars[] is created by scrambling the input registration key according to the permutation given in RegistrationGate.decoder. Next, the chars[] array is not allowed to have the digits 3, 6, or 7 in the last six positions. But after this check is performed, E is replaced with 3, C is replaced with 6, and A is replaced with 7. Therefore, the above set of keys produces the following sequence of values for n at the end of RegistrationGate.check: 995, 996, 997, 998, 999, 1000, 1001, 1001, 1002, 1003, 1004, 1005, 1006, 1007, 1008, 1008.

At the end of this check() function, we first call isInBounds(n). This compares our value of n to RegistrationGate.maxN. If our value is smaller, then we return true. Otherwise, the function returns false, after RegistrationGate.maxN is increased by 1 if the two values are equal. We need to pass this check for increasingly large values of n, which is why we increase n by 1 in each step. Then we have to pass the second check, which is true when n is a multiple of 7. This

explains why we repeat the keys in the above sequence that correspond to values of n that are multiples of 7. We then reach the third check, which calls the function checkCore. This is where our runtime increases dramatically. The second time that we input the key 000000101000, the server response takes a few hundred. The second time that we input the key 000000801000, the server response takes over an hour. A benign user cannot view their ballot while this attack is running, because their registration key cannot be checked. Therefore, this attack is successful."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The exploit provided by Utah during the take-home exceeded the input budget; however, an exploit is not required, and an exploit generated during the take-home exercised the same vulnerability but was crafted to satisfy the input budget.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"The attack we had found (which is described below) is correct, and is essentially the same we used, but doesn't fit in the budget. After discussing it with the other teams we realized our original attack was not in budget. Together we determined that a smaller set of keys would work (8 vs 16):

979497500901
979497C00901
979497A00901
979497800901
979497900901
079407001901
094109101009
094109101009

This alone is not enough to get it in budget, so you also have to use a technique to reduce the SSL overhead. Using a Python script, we were able to send multiple HTTPS requests across a single SSL connection. This reduces the overhead of SSL setup to be paid only once instead of for each registration key. The total budget used is around 3KB and the times measured by every team are around 740 seconds for the whole script. This was tested using a script developed by Northeastern in collaboration with us and the other teams. The timing and input budget used come from experiments we did using it. […]"

Live-Collaborative Evaluation:

Utah's exploit appears similar to the one proposed by Northeastern during the take-home. The team noted that they collaborated with all the R&D Teams to reach this exploit.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.10 SimpleVote 4

A.6.5.2.10.1 Question 030 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

Same as Question 038.

Take-Home Evaluation:

The challenge contains an intended vulnerability that enables an attacker to erase the cache used to process registration keys. This potential unintended vulnerability reported by the team exceeds the input budget.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“The attack we found was not in budget although it was in the correct place. Both Colorado and GrammaTech found the in budget attack by manually fuzzing the possible inputs looking for a smaller attack. Colorado was suspicious of this code area because of information provided by their tool Code Hawk. The attack they found consists of submitting the series of registration keys below:

900090500000
900090500000
900090500000
Because they use only 3 messages to trigger the slow behavior the attack is well in budget, which they verified. The attack uses a trapdoor in checking the registration keys. Before a key is checked in the database, it is first processed by the SanctionAuthenticator. This authenticator includes a trapdoor that can be triggered under a very specific set of parameters. The pattern of registration keys was chosen carefully to pass through several conditions in SanctionAuthenticator.confirm. First, the array chars[] is created by scrambling the input registration key according to the permutation given in SanctionAuthenticator.decoder. Next, the chars[] array is passed through confirmExecutor, and is not allowed to have the digits 3, 6, or 7 in the last six positions. But after this check is performed, E is replaced with 3, C is replaced with 6, and A is replaced with 7. Therefore, the above set of keys produces the following sequence of values for n at the end of SanctionAuthenticator.confirm: 995, 955, and 955 Eventually the SanctionAuthenticator calls confirmValue which the ChaoticOperator class and it is this class which does all the work. Each 955 executes a different part of the function, first priming, then erasing elements from the cache and the final 955 forces it to recalculate a lot of values which is what takes up the resource usage.”

Live-Collaborative Evaluation:

Utah worked with all the other R&D Teams to identify the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.11 STAC Coin

A.6.5.2.11.1 Question 007 (AC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“The function Ledger::addTransaction includes a nested for loop. It loops first over all inputs for the transaction, and then compares each input to each unspent output. So to make this run for a long time, we first need to create many unspent outputs. With 1 MB of space, we should be able to create several thousand of these. Then with the second MB of space, we create a transaction with several thousand different input addresses. That may be sufficient to cause this double for loop to take more than 100 seconds, so we give this question a tentative yes. If that's somehow insufficient, it may also take some time to verify the signatures for all of the transactions. It takes around 8.5 milliseconds to verify a 256-bit ECDSA signature:

https://crypto.stackexchange.com/questions/3216/signatures-rsa-compared-to-ecdsa. If we can send 10,000 transactions, that alone would take around 85 seconds, which is close to the limit.

Future: Test both of these attacks on the NUCs."

Take-Home Evaluation:

The team's response appears to be an argument for a vulnerability but responded not vulnerable. The challenge does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Iowa found a potential exploit by creating a malicious peer which doesn't respond to certain requests. Unfortunately, this requires the client server to repeatedly request (200 times) from a malicious server and we think this is out of scope (and improbable.) In their potential attack the attacker uses a malicious server that doesn't behave as expected. When the malicious server receives some request it hangs and never responds. The benign client server doesn't have a timeout on the operation so it waits forever. If the client repeats this 200 times (the amount of available threads in the client spark server thread pool) it uses up all of the threads available to handle requests. Finally, when the benign request comes in there are no threads available so the resource usage limit is exceeded. This seems like a promising vector but it is not clear how you can force the benign client to make the 200 requests. For the rest of the application we don't believe there is another attack. We investigated the function Ledger::addTransaction, which includes a nested for loop. It loops first over all inputs for the transaction, and then compares each input to each unspent output. So to make this run for a long time, we first need to create many unspent outputs. With 1 MB of space, we should be able to create several thousand of these. Then with the second MB of space, we create a transaction with several thousand different input addresses. It may also take some time to verify the signatures for all of the transactions. Considering all these pieces there might be an attack but we don't believe it is in budget."

Live-Collaborative Evaluation:

Utah collaborated with Iowa, Vanderbilt, GrammaTech, and Northeastern to correctly conclude that the challenge does not contain an intended vulnerability. If required further analysis may be performed by Apogee to confirm that Iowa's exploit does not fit within the budget.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.11.2 Question 008 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"The only time the program creates a file is when it writes a wallet to the wallet.dat file. So we just need to check how much storage a wallet will need. […] Each key is 256 bits. This doesn't seem like nearly enough of an increase; elliptic curve key sizes are fairly small. […] The depth map has one entry per block, so it only adds a constant factor times the number of blocks.

Each block stores a nonce, a previous block hash, and a list of transactions. This looks like a dead end for an attacker, since they would have to create a list of all of the transactions.

[…] None of these seem to increase significantly in size above the size of the input transactions. So our answer for this question is no, there does not seem to be any space attack satisfying these requirements.

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“We stand by our original [answer] no. GrammaTech has an interesting attack idea which we believe doesn't follow the spirit of the question. Their idea is to generate a long (more than the resource usage limit) blockchain offline (creating transactions, mining, creating transactions mining, etc...) and then notify the client. It will then ask for the blockchain from the attacker who (crucially) responds (they argue this doesn't count towards the input budget because it is a response not a request) with the whole large blockchain which is then written to the [client’s] wallet. We think this does count to the budget and so isn't a viable attack. Below is our original answer which analyzes the rest of the application. The only time the program creates a file is when it writes a wallet to the wallet.dat file. So, we just need to check how much storage a wallet will need. We need to find an input that causes the space used to be over 1000 times larger than the input size. A wallet contains a BlockChain and a list of key pairs with public and private keys. Each key is 256 bits. This doesn't seem like nearly enough of an increase; elliptic curve key sizes are fairly small. A BlockChain contains a list of blocks, a ledger, and a depth map. The depth map has one entry per block, so it only adds a constant factor times the number of blocks. Each block stores a nonce, a previous block hash, and a list of transactions. This looks like a dead end for an attacker, since they would have to create a list of all of the transactions. The ledger has:

- A transaction map, which maps ID's to transactions.

- A map from user addresses to balances.

- A set of unspent outputs, each of which contains a transaction ID and an index.

None of these seem to increase significantly in size above the size of the input transactions. So our answer for this question is no, there does not seem to be any space attack satisfying these requirements.”

Live-Collaborative Evaluation:

Utah collaborated with all other R&D Teams to correctly conclude that the challenge program is not vulnerable. We agree with the team’s assessment of the proposed exploit by GrammaTech as exceeding the spirit of the question in the calculation of the required input budget.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.11.3 Question 042 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“[…] The random number generator is the built-in SecureRandom class. Once an amount has been spent, it is removed, so replaying a transaction is also not possible.

It seems, then, that the only way to spend another user's coins is to acquire their private key. One way to do this would be by guessing their wallet password, and sending a request to /wallet/:walletPassword/export, which outputs all of the key pairs.

However, if the user chooses a password correctly as a random eight character string, this would be infeasible. And the time taken to check the hash of a user's password when they log in does not depend much on what password they use, because the program deliberately spends extra time computing various constants.

The space used likewise seems to depend on the transactions themselves, but never on which keys are used. So there seems to be no side channel attack, assuming the security of the underlying cryptographic libraries."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.12 Swappigans

A.6.5.2.12.1 Question 002 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"There is a complexity vulnerability in time that can cause the real runtime of a benign login request to exceed the budget of 6 seconds. When a user logs in to Swappigans, the full item list is populated. If enough items have been added to the list in advance of the login event, the result is delayed significantly. This is because the code that generates the HTTP response has to accumulate the listings for each item in the inventory. This is done using a while loop wherein a result string has each item's listing HTML appended to it. This is a slow process."

Take-Home Evaluation:

The server is provided as-is with the provided canned data. This challenge contains an intended vulnerability and a potential unintended vulnerability. Both vulnerabilities are associated with adding items to the list and attempting to make a purchase. The potential vulnerability highlighted by Utah appears to be when the list of items available for purchase is sent to the user. We do not believe there is a sufficient input budget to make this list sufficiently long enough to delay the benign request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"There is a complexity vulnerability in time that can cause the real runtime of a benign login request to exceed the budget of 6 seconds. In Swappigans when we bid on an item the app will try to find items in our bucket that are closer to the value of items we are bidding on. To do this the app goes in a recursive call in SwappigansItemMatcher class. The important method here are calc which in turn calls the apxSumRecursion to find item which are closer to the value of item to be purchased. For the actual attack we should add the items in increasing order or decreasing

order increasing by 2 times or decreasing by half. Bid on item that to be purchased it will go in a recursive call. This application is Single threaded when the benign user tries to log in they will not be able to enter till the recursive call finishes."

Live-Collaborative Evaluation:

Utah collaborated with Colorado, GrammaTech, Iowa, and Vanderbilt to identify the same unintended vulnerability identified by other teams during the take-home engagement.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.12.2 Question 020 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"[…] To generate this key the application uses SessionTokenManager which in turn uses a random number for which the seed is set as System.currentTimeMillis()/1000L. If this seed can be guessed, then the entire session id system can be compromised. Since the attacker is allowed to create an account and login, the attacker can test whether a given input is correct by looking at whether a guessed key results in the session id given by the server for an attacker's account. If the server has been started within the last year, there are only 31 million (365*24*60*60) values to try. This is well within the capability of a modern computer. If checking each guess takes 10^5 cycles, a 1GHz machine can test 31 million values in under 53 minutes.

Once the correct key is discovered, a session token can be forged for any user, and the the attacker can fire a purchase request using the target username and the session token."

Take-Home Evaluation:

Utah flagged the same unintended out-of-scope vulnerability as several other teams. A key assumption of this vulnerability is the attacker knowing when the server was started. In future challenges we will work to clarify that the attacker does not know when the server was started.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.12.3 Question 031 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Once a purchase is complete, the item purchased is not recorded by the application, and thus it cannot be discovered using any request to the application. The passive operation is thus they only way there could exist a side channel.

Thus, we consider the passive operations. The attacker can observe how long the purchase item request for a user take. The item purchase flow will take variable time depending on the price of the item being purchased, as it will try to match it with an item that is of lower value that the item being purchased and it has to be part of items listed by the user. (The class of interest here is

SwappigansItemMatcher). Since the attacker does not have control over the list of item and prices of the benign user passive operations cannot be used to determine the item purchased."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable. Note, for these types of questions the provided canned data is a part of the analysis the canned data's public items of things available for sale is known to the attacker. It is unclear from the response if this was known by the team.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.5.2.13 Tollbooth

A.6.5.2.13.1 Question 011 (AC Time, Intended Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"[…] After examining the different kinds of packets, we conclude that the main factors in the running time are the number of transponder ID's and the number of leaf nodes. [… timing charts are included…] By creating M leaf nodes and registering N new transponder ID's, an attacker can send data of size $O(M + N)$, and force the root to send data of size $O(MN)$. Given the input budget, this is not a sufficient blowup to cause the root to exceed the time budget."

Take-Home Evaluation:

Utah incorrectly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"There doesn't seem to be a way of keeping the root server running, and have the runtime of the benign request exceed its budget, and doing an attack. GrammaTech which answered yes to this question before the live engagement had discovered, by looking at the code, that if a carPassed packet with a null destination is sent it would be broadcasted across the network. They hadn't tried it and during the live engagement they did. Unfortunately, the result is not what is expected, there is an "equals" check on the destination which crashes the root server due to a null pointer exception. As the root crashes when the leaf node does its benign request it doesn't get the response back. So, this could be an attack, but we think it not in scope. What follows was our original answer which didn't consider the above, and considers the root uncrashable. After examining the different kinds of packets, we conclude that the main factors in the running time are the number of transponder ID's and the number of leaf nodes. The following chart shows how the root and the leaf nodes respond to each operation.

[…] The next chart shows how the root node responds to requests by a user through the web interface: […] By creating M leaf nodes and registering N new transponder ID's, an attacker can

send data of size $O(M + N)$, and force the root to send data of size $O(MN)$. Given the input budget, this is not a sufficient blowup to cause the root to exceed the time budget."

Live-Collaborative Evaluation:

The challenge contains an intended vulnerability; however, the vulnerability discovered by GrammaTech achieves the required goal of the challenge question in that it by crashing the thread, the benign request to be delayed does not receive a response.

Post-Engagement Analysis Ruling: Incorrect

A.6.5.2.13.2 Question 024 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"[…] When a new node is created, the root sends its own UUID, and adds the public key for the new node's UUID to the routing table. It does not perform any error checking on the UUID, but assumes that it is generated randomly. An attacker can use this to hack the routing table. He first creates a new leaf node and receives the root's UUID. Then he creates a second new leaf node and sets its UUID to be the same as that of the root. This causes every node to replace the root node's public key with the attacker's public key in their routing table. The attacker can then read all of the messages between the benign leaf nodes and the root. However, the leaf nodes do not have access to the passwords for the users; only the root node does. So, this is not sufficient for the attacker to log in to the target user's account and transfer funds to himself."

Take-Home Evaluation:

Utah correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

No Change.

Live-Collaborative Evaluation:

No Change.

Post-Engagement Analysis Ruling: Correct

A.6.6 Vanderbilt University

A.6.6.1 Vanderbilt Overview

Vanderbilt answered the third most questions in the take-home engagement (40), and achieved the tied third highest R&D Team accuracy (73%). Iowa, GrammaTech, Colorado, and Vanderbilt all achieved between 73% and 76% accuracy in the take-home. Accounting for the heavy bias towards non-vulnerable challenge questions in this engagement, the TPR and TNR values provide a clearer view of the strengths and weaknesses of the team's capabilities than the accuracy. Vanderbilt had the highest TNR of all R&D Teams (96%), but the second lowest TPR (38%). These metrics indicate that strength of Vanderbilt's approach is in helping the team rule out vulnerabilities, but the team has difficulties with identifying vulnerabilities.

Vanderbilt had the most difficulty with AC Space questions (38% accuracy) and achieved above an 87% accuracy in all SC question categories. In the segmentation of all questions (unanswered

questions assessed as incorrect), the team achieved the second highest R&D Team accuracy, tied with Iowa State. Vanderbilt's TPR remained the same in the segmentation of all questions, but the team's TNR decreased to 85% - the second highest R&D Team TNR.

During the live-collaborative engagement, all R&D Teams achieved between a 97% and a 100% accuracy. Vanderbilt achieved a 98% accuracy, tied with Northeastern and GrammaTech. Collectively, the R&D Teams identified 3 unintended vulnerabilities during the live collaborative engagement – Railyard Question 013, Class Scheduler Question 017, and SimpleVote Question 038. Railyard Question 013 and Class Scheduler Question 017 came out of a collaborative effort from all teams. The SimpleVote Question 038 unintended vulnerability was confirmed with a clever exploit by Northeastern. The only question that the R&D Teams collectively got incorrect was the intended SC Space vulnerability in Case DB Question 019.

A.6.6.1.1 Take-Home Engagement Scores

Table A-137: Engagement 6 Vanderbilt Take Home Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	7	88
SC Time	11	10	91
SC Space/ Time	3	3	100
AC in Space	8	3	38
AC in Time	10	6	60
Total	**40**	**29**	**72**

Table A-138: Engagement 6 Vanderbilt Take Home Engagement Vulnerablity Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	16	6	38
Not Vulnerable	24	23	96
Yes Answer	9	7	78
No Answer	31	22	71

A.6.6.1.2 Live-Collaborative Engagement Scores

Table A-139: Engagement 6 Vanderbilt Live Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	7	88
SC Time	11	11	100
SC Space/ Time	3	3	100
AC in Space	8	8	100
AC in Time	10	10	100
Total	**40**	**39**	**98**

Table A-140: Engagement 6 Vanderbilt Live Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	16	15	**94**
Not Vulnerable	24	24	**100**
Yes Answer	16	16	**100**
No Answer	24	23	**96**

A.6.6.2 Vanderbilt Specific Responses

A.6.6.2.1 BattleBoats 3

A.6.6.2.1.1 Question 022 (AC Space, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"The Jackson parser has some loops and recursions, but they seem to be well-constrained loops. ItemList.split, for instance, breaks up a String s by some delimiting String sp and places those substring entities into the List append. It terminates when the delimiter is no longer found in the string. The JACKSONObject.toJACKSONString method iterates over a HashMap of entries to pull all the objects from Map and add them (comma-separated) to a StringBuffer. The loop iterates for the number of entries in the HashMap and does not modify the Map during the loop. All of these entries seem to be very well behaved. Other potentially recursive routines are the message handler inner classes in proto/Battleboats. […] One of the interesting loops found was HitPinpointer.determineHitTime that uses BigDecimal calculations in a somewhat open loop and is called for each shot taken by the user. This is of interest because the MathContext (or max number of digits) of the BigDecimal values are 10000. For a 1000 MB memory limit, this would require around 100k entries to achieve this if the values are of the maximum length. The loop iterates the step method for the tCurrent and tPrevious values where tPrevious is initialized to 0 and tCurrent to 1E-100. […] Each iteration after the 1st 25 will place the tCurrent value and its corresponding height value in the Map trajectory, which then can potentially become very large. However, it is a local variable, so once the calculation has completed the allocation can be garbage collected. […]"

Take-Home Evaluation:

Vanderbilt did not identify the intended vulnerability by which a hash table can be used to exceed the resource usage limit during shot computation.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Grammatech and Colorado were able to exploit the challenge program so that memory usage exceeded 1GB. They based their exploit on the same method we thought was suspicious, HitPinpointer.determineHitTime. Like us, based on the code, they theorized it was not possible for the challenge program to exhibit memory usage > 1GB. In practice, however, this was not the case. They found that it was possible for the challenge program to exhibit such high memory usage, while being under the budget. Their attack is based on sending three consecutive shots with extremely small floating point numbers for the height and velocity of the shot. They shared the attack scripts with all the teams and the exploit does seem to be replicable. The attack scripts are attack.sh (to be run by player1) and attacker-ships.txt (also for player1). The “magic” numbers they provided were generated by their fuzzer. We believe we could also generate these numberswith our tool Kelinci-WCA but were unable to do so during the live engagement due to timing constraints.Additionally, Iowa found what they thought was justification for why the spike in memory usage is seen. The particularly suspicious code involves thesqrt calculation in the HitPinpointer.determineHitTimemethod, which uses the “magic” velocity number given by Colorado’s fuzzer. Due to the way doubles and BigDecimals are handled within the code, the very very small velocity number caused BigDecimal’s to take up significantly more room during the computation. Since these aren’t garbage collected immediately, it causes the memory to spike significantly, above the 1GB mark.”

Live-Collaborative Evaluation:

Vanderbilt identified the intended .

Post-Engagement Analysis Ruling: Incorrect

A.6.6.2.1.2 Question 026 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“During the initialization phase, a player can issue the following commands […] Of these commands, only accept_game, decline_game and end_placing_boats actually send any information over the network. Both decline_game and accept_game contain no information regarding boat placement and can be disregarded from the analysis. Once a player issues the end_placing_boats command, EndPlacingBoatsCommand.executeService() sends a message to the opponent, which contains some information regarding the location of the opponents cannon. It is unclear how an attacker could use this information.

The only other active operations an attacker could perform is during the game itself, namely through the shoot command. Using JAnalyzer gives us the following chain of events once the shoot command has been issued by the attacker: […] The determination of hit squares is done in OceanScreen.obtainHitSquares(). First, the necessary physics calculations are done to obtain the actual squares of where the shot landed. Then the shots are checked against all the squares of the boats on the ocean screen. For each square, either a MISS is calculated, or single boat was HIT. Regardless of whether a boat was hit, these squares are then shifted translationally to radar

screen coordinates, and sent back to the attacker (assuming the attacker made the shot). Aside from the basic information of whether or not a boat was hit (which is available through the main channel), no information regarding the ocean coordinates of boats is leaked. Since this is the only other operation an attacker could perform, it does not seem that there is a space side channel."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.1.3 Question 040 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"[…] only accept_game, decline_game and end_placing_boats actually send any information over the network. Both decline_game and accept_game contain no information regarding boat placement and can be disregarded from the analysis. Once a player issues the end_placing_boats command, EndPlacingBoatsCommand.executeService() sends a message to the opponent, which contains some information regarding the location of the opponents cannon. It is unclear how an attacker could use this information.

The only other active operations an attacker could perform is during the game itself, namely through the shoot command. […] The determination of hit squares is done in OceanScreen.obtainHitSquares(). First, the necessary physics calculations are done to obtain the actual squares of where the shot landed. Then the shots are checked against all the squares of the boats on the ocean screen. For each square, either a MISS is calculated, or single boat was HIT. Regardless of whether a boat was hit, these squares are then shifted translationally to radar screen coordinates, and sent back to the attacker (assuming the attacker made the shot). As each boat's squares are checked to see if they were hit (using a standard Java map, containsKey), the time it takes to determine which boats were hit does not seem to leak any timing information regarding the individual ocean coordinates of each boat. Thus, there does not seem to be a timing side channel here."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.1.4 Question 043 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"[…] Of these commands, only accept_game, decline_game and end_placing_boats actually send any information over the network. Both decline_game and accept_game contain no information regarding boat placement and can be disregarded from the analysis. Once a player issues the end_placing_boats command, EndPlacingBoatsCommand.executeService() sends a message to the opponent, which contains some information regarding the location of the opponents cannon. It is unclear how an attacker could use this information.

The only other active operations an attacker could perform is during the game itself, namely through the shoot command. Using JAnalyzer gives us the following chain of events once the shoot command has been issued by the attacker […] The determination of hit squares is done in OceanScreen.obtainHitSquares(), which simply calls OceanScreen.pullOceanSquares() […] The attacker, being one of the players, does not know where their cannon is located on the opponents board. One possible way an attacker could narrow down the set of locations their cannon would be is to try shooting in different directions with the goal of gauging how far away from the edge of the game board their cannon lies. From the logic in pullOceanSquares(), if the square the cannon shot lands off of both the ocean board AND the radar board, then pullHits() is not called, indicating a clear timing difference. However, if the shot lands outside the ocean board but within the radar board, pullHits() is invoked. Thus, the attacker can determine through a timing ifference if they shot off the radar board, but not necessarily if they shot off the ocean board.

The board can range from 15x15 to 20x20 in size. Thus, in the worst case of a 20x20 board, the cannon could be in any of the 400 squares. The radar board essentially doubles the dimensions, creating a 40x40 board size, with 1600 squares. It is not clear that the timing differences from a shot on vs off the radar board would be allow an attacker to learn the position of their cannon, especially given the operational budget of only 100 operations."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.2 BraidIt 3

A.6.6.2.2.1 Question 004 (AC Space, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"The SLF4J Logger is used again, which is used to generate the log file staclog/stac.log. A maximum log file of 10.5 MB can be produced, at which point the file is renamed by adding a

digit to the end of the name and a new stac.log file is started. Up to 3 additional files can be generated for a total of up to 42 MB of file space, which is above the resource usage limit. Only the INFO level of logging is enabled, but each command that is issued generates a logfile message of around 120 bytes each.

The important thing is that the braid modification commands do not transfer any data on the network. After the braid modifications have been made, issuing the "send_modified" command does send a network message. The following table lists the available commands and their approximate impact on the input budget and the effective output produced. […]

Since the input payload budget is only 2048 bytes and the connect uses up 1037 of them, only 1011 bytes are left in the budget for the game itself. This precludes attempting to use the decline_game or disconnect for attempting repeated games or connections, since they would very soon eat up the input budget. Also, the print, help, history and exit commands do not produce any output, so there is no use in using them either. However, this still allows unlimited use of the modification commands (listed in italics above). Slightly more data output can be generated per command if DEBUG level logging can be enabled, but there doesn't seem to be any provision for that.

The question does not imply the location of the resource usage limit (on the victim or the attacker), so it would seem that ANY log output is considered part of the usage limit. That being the case, the 2 new commands that were added, insert_identity and collapse_identity) perform inverse functions on the braid. That is, given a braid, by performing insert_identity on it will add a matching pair and the collapse_identity will remove one (possibly not the same, but the braid size will remain the same as it was before). Therefore, in repeating a loop of inserting and removing the matching pairs one could perform this indefinitely to generate as large a total file size as necessary (up to the limit) without exceeding the budget. Yes, there is a vulnerability by an attacker iteratively issuing the insert_identity and collapse_identity prior to sending the send_modified command to the victim."

Take-Home Evaluation:

The challenge contains an unintended vulnerability reported by Iowa and Two Six Labs. The potential exploit reported by Vanderbilt appears to depend on running the attacker and target user applications on the same physical machine. We believe that when this exploit is run on the reference network environment, it is the attackers resources that will be consumed.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"Iowa presented another attack using the provided Pythonscript where they sent a special string to the benign user in which the comparison routine run in benign user's clientside runs for a long time and logs a lot of operations. They found it by searching the string using the constraints

given in compare function, their attack and search code is attached. We could replicate this discovery by fuzzing the string with KelinciWCA but we didn't realize we could change the sent string and exceeded the limit on attacker's side as the question doesn't clarify which side the limit has to be exceeded. The question doesn't clarify on which side we are trying to store files, so our findings are correct as well, but this creates logs also on benign user's side and exceeds the resource usage limit on that side as well"

Live-Collaborative Evaluation:

The unintended vulnerability identified by Iowa and accepted by Vanderbilt was confirmed.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.2.2 Question 027 (SC Space, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

“We've ran the full game where for each round each secret (selection of braids from 1 to 5) is run 250 times and monitored the network traffic with Profit. To determine which braids are selected by the opponent, we focused on packets transferred on those rounds because they are directly related with it and the code shows that that's where the modified braid is sent through the network. Between runs, only modified braid packet is changing, so we focused on that and used Profit to find information leakage through the network. The secret distribution and values depicted are shown below where the x axis is the value of the observable, y axis is probability of secret being a value among secrets. […]

The initial entropy of secret is 2.321 bits (log5) and for first round 0.001 bits are leaking, for the second round 0.006 bits are leaking when we model the observable distribution as a Normal distribution. Therefore, we can't learn anything from observing the packets and we need to guess randomly to guess the secret. Assuming 8 operations to play the game, 4 operations aren't enough to guess the secrets in 2 played rounds.

If we don't observe blindly and take the actions the attacker can do into account to alter the side channel, we have 2 actions that can affect the game. Those are setting the seed from attacker's side and setting number of strands. Setting the seed on attacker's side doesn't seem to affect the benign user's braids in length or patterns of letters, so this action doesn't gain us any information. Setting the number of strands affect the characters used and some operations that benign user can take have different conditions for different characters but we'll show that those don't affect our information gain as well.

Number of strands affects the unique letters in strings but all the length altering operations are allowed with min to max number of strands. Operations are expand3, expand5, insert_identity, collapse_identity, swap and triple swap. Swap and triple swap doesn't change the lengths but expand3 and insert_identity increases the length by 2, expand5 increases the length by 4, collapse_identity may decrease the length by 2. Also the lengths of braids are uniformly distributed and not influenced by seeds so we can only distinguish even length strings from odd length with these operations. In worst case, if there are 5 even or 5 odd braids (which is a possibility that happens 6% of the time), whatever the benign user selects and modifies, we can't distinguish it from others. Lastly, before the braids are sent, the selected braid is formatted using Java's String.format() in PlaitSelectedPhase.takePlaitString(). This string formatting has a padding option but the padding applies if the padding is longer than the string and sets the string length to the padding length, only filling the remainder. The padding size is selected as string's length in the aforementioned function, so there's no remainder and the length of the string stays the same without any modification dependent on selection. Therefore we don't gain any information from using main channel information and side channel packet size in the same time and we can confidently claim there's no leakage within the given budget in the worst case.”

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.3 BraidIt 4

A.6.6.2.3.1 Question 032 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"[…] The initial entropy of secret is 2.321 bits (log5) and for first round 0.461 bits are leaking, for the second round 0.446 bits are leaking when we model the observable distribution as a Normal distribution. We can learn some information when we observe the packets (The secret marked with blue has lower packet size values than the rest.) but we can't infer the full secret in average. Assuming playing the game takes 8 active operations, 4 operations to use the oracle to guess the secret doesn't seem to be enough for secrets of both rounds.

As we noticed, the selection index is somewhat correlated with the size of the feature, so we investigated how the braids are sent over the network. We noticed that before the braids are sent, the selected braid is formatted using Java's String.format() in WeaveChosenState.obtainWeaveString(). This string formatting has a padding option but the padding applies if the padding is longer than the string and sets the string length to the padding length, only filling the remainder. The padding size is selected as the selected index multiplied with selected string, so the actual padding is (index-1)*len(modified string). There's also a 71-72 bytes overhead observed on the network. Using this information, we can observe which index the opponent chose. The problem is that before the strings are sent, they are shuffled, so as an attacker, our representation of mapping between indexes and strings are different. Even if we know that opponent chose a particular index, the string corresponding to that index may be different from what the opponent chose. The shuffling doesn't seem to be leaking and it seems to be random, therefore we can guess what index the opponent chose but not the string itself.

SelectionsState.prepareForSelections() seems to shuffle the strings before showing it to the attacker and if we could modify and recompile the clientside code that the attacker uses, so that this shuffle function did not execute, the exploit we mentioned would work to determine the braid the benign user selects. Assuming we can alter the clientside code, we can state that we can determine which braids are selected using the side channel information. If our assumption is wrong (we can't alter the clientside code), then we can only obtain the indices of the strings that the benign user sees and that's not enough to leak the secret information."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable. However, the team noted a potential vulnerability in the randomization of received strands prior to displaying to the attacker. The randomization of the strands prior to being sent is believed to prevent the potential vulnerability rejected by Vanderbilt.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“Northeastern, and GrammaTech teams had the malicious attacker idea where they modify the attacker’s clientside to prevent shuffling and measuring the network to infer the string, same as us. Colorado had the index analysis but we showed that you would needto obtain the string to answer the question correctly, not just the indices. We collaborated with Iowa team to test the attack where we comment out the shuffle function in SelectionsState.prepareForSelections() on client code of the attacker and rebuild it. They noticed that the strings are shuffled even with commenting that out and that function is also called in the victim’s code to have another index representation list called weavePointers and because of that shuffle, it doesn’t matter if we shuffle orrefuse to shuffle the list. Grammatech proposed learning the seed of generation of strings, so that they can reconstruct the sequence of strings and created an exploit which does the prediction given the strings. Because all construction uses the same RNG(Random Number Generator), and the order of strings that the user sees and the generation of strings are in the same order, we can use the information of generation of strings to predict the next sequence of characters. The strings are generated in function Weave.grabRandomWeave() and what it does is for each character, it chooses one index from the list of unique characters randomly with command Weave.random.nextInt() and it randomly chooses a boolean value to make it capital or small letters. The number of unique characters is determined by the number of strands (number of unique characters in braid) which the attacker chooses when they propose the game. GrammaTech implemented an attack which takes a sequence of strings and provides the next set of characters given the determined seed using a modified version of JavaCG and a way to translate the character values to the indices which you can do because they are ordered in the same fashion each time you run the program. The attack can handle strings over15 length to do correct predictions but we can stitch the strings after we know a partial sequence as well or try different combinations of shorter strings to predict next characters which will not use the budget.

We believe using this attack, you can reconstruct the order and from the side channel information giving the index, you can learn which string the user chose for both rounds.”

Live-Collaborative Evaluation:

GrammaTech’s Exploit to predict the random number sequence was confirmed as a valid but out-of-scope vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.4 Calculator 1

A.6.6.2.4.1 Question 014 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

“We are not explicitly given the expression grammar that is accepted by the calculator program in the questions. We thus need to discover the grammars permitted by the calculator, i.e., the program terminates normally given any inputs formed by the grammars. We start with an infix grammar, i.e., E := (E)|E+E|E−|E∗E|E/E|E^E|ErE, and subsequently discover other grammars permitted, e.g., postfix grammar, etc. We do this by first using ANTLR [3] to automatically generate an expression parser for the infix grammar as described above. Note that ANTLR takes as input a grammar file from user, and automatically generates a parser for that grammar. We then use Kelinci [1], a fuzzing tool, to fuzz input expressions in the form of infix grammar. For example, an input expression (1 + 2) was given to Kelinci. Kelinci then tries to fuzz the given

input to generate various other different inputs. Note that inputs generated by Kelinci are not necessarily accepted by the calculator program. Any inputs generated by the fuzzer that are permitted by the calculator, but not permitted by the infix-grammar parser generated by ANTLR, would be of our interest. This way, we can systematically discover other grammars/rules permitted by the calculator other than the infix grammar. The result of this step shows that the calculator program accepts also postfix grammar, e.g., E := (E)(E)+, and more interestingly, it accepts the following grammar rule: E := (E)(E)(E) ..., in which the element (E) can be repeated by any number of times.

Given the grammars discovered in the Step 1 above, we now try to generate inputs that can either trigger the vulnerabilities in questions. We implemented our own expression fuzzer that works on the permitted grammars loosely allowed by the calculator. The reason for implementing an expression fuzzer by ourselves hinges on the fact that general fuzzing tools like Kelinci [1] generate too many invalid inputs that are not allowed by the calculator, which subsequently cause the program to terminate early on without consuming much memory or running time. Our fuzzer is implemented on top of the ECJ evolutionary framework [2]. It extends ECJ with several mutation operators tailored for mutating expressions conforming to the permitted grammars. It implements an objective function, which favors candidates that consume either more memory or running time, to facilitate genetic-programming based fuzzing style.

We run the fuzzer on small budget (less than the input budget allowed) to discover whether there is any input that can potentially trigger the vulnerabilities in less than 2G of memory or 60 seconds. The results show that an input of the form: (E)(E)(E) ..., wherein E is: 999^1695*999^1695*999^1695*999^1695*999^1695 would cause the program to run slowly. We then manually confirmed that the input of the above form with (E) repeated 212 times (in the budget of 10KB that is allowed by the program) would cause the program to run longer than 120 seconds. The fuzzer were not able to discover input that cause the program to consume more than 4G of memory. We elaborate reasons for that below, based on our manual investigation further into the program. The program allows the rule of the form (E)(E)(E) ..., and in fact, this causes it to keep the results of each (E) on a stack until the program finishes. This means that the memory consumed by the program is linearly proportionate to inputs of this form. This signifies a promising direction for generating inputs that trigger more memory consumption. However, our investigation shows that this will not help us exceed 4G of memory given the input budget of 10KB. That is, we instrument the stack holding all expression E, and discovered that the most memory that an expression E consumes is 20KB. In order for the heap memory to exceed 4G, we then need the stack to contain at least 209715 items of expression E kept on the stack. Now consider the input budget of 10KB, to make an input with at least 209715 items of expression E in the form (E)(E)(E) ... above, we need to represent each expression E by: 10240/209715 (less than byte), which is impossible. Thus, we cannot count on input of the above form to trigger more than 4G of memory usage. We note that this proof is sound, but not complete, in the sense that we did not discover other input grammars that can allow us to exploit the memory, other than the above input form."

Take-Home Evaluation:

The challenge did not contain an intended vulnerability; however, several teams discovered an unintended vulnerability. Vanderbilt's response makes an incorrect assumption about the application's input budget. While the application does attempt to limit inputs $\leq 10kB$, the attacker can use the $60kB$ budget to perform a multiround attack if indeed all inputs of $\leq 10kB$

are properly filtered. It is unclear from the response if the team meant the $10kB$ limit as a restriction on the single inputs to be considered.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After discussions with Colorado and Iowa State, we realized there is an exploit. As described above in our original justification, our fuzzer generated an input ("999^1695*999^1695*999^1695*999^1695*999^1695") that, when repeated 212 times, caused the program to run slowly, but did not cause the program to exceed to memory budget. Interestingly, Colorado's fuzzer revealed a very similar but shorter version of the input ("999^1695*999^1695*999^1695) that, when submitted as an input request to the server 60 times with a very short pause in between each request, will exceed the memory budget.

Also, for this application and similar future programs we can consider the number of queries and also gap which can be received from the Kelinci (Our fuzzing tool) and use them for detecting these kinds of bugs. Our tools has this ability to detect these kinds of vulnerabilities if we consider it on a driver."

Live-Collaborative Evaluation:

Following collaboration, Vanderbilt was convinced to change their answer to a correct Yes.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.4.2 Question 016 (AC Time, Unintended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"Please refer to detailed analysis in Q14. The report here only shows the input that triggers the vulnerability. All analysis steps done for this question are the same for Q14.

The results show that an input of the form: (E)(E)(E) ..., wherein E is: 999^1695*999^1695*999^1695*999^1695*999^1695 would cause the program to run slowly. We then manually confirmed that the input of the above form with (E) repeated 212 times (in the budget of 10KB that is allowed by the program) would cause the program to run longer than 120 seconds."

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. The potential unintended vulnerability reported by Vanderbilt was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.5 Calculator 2

A.6.6.2.5.1 Question 003 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"The only place where a log file is written is inside the exception handlers of the overridden handle() method of the AbstractHttpCoach class. Two kinds of exceptions are InterruptedException and ExecutionException, though in the case of the ExecutionException the log file will only be written to if the exception is not an IOException. We investigate how to throw such exceptions, but do not find a way to trigger them.

1. An InterruptedException is thrown if the thread performing the execution of this.doHandle(httpExchange) is interrupted during the execution. However, nowhere in the project code is there a call which will cause the interrupt of an executorService's thread, so the logging associated with an InterruptedException cannot be reliably triggered.
2. ExecutionExceptions occur when some error occurs during the execution of a computation. However, the doHandle method only throws IOExceptions. As such, any caught ExecutionException will be an instance of IOException, and therefore the log file will not be written. We also tried to disconnect after just connected, but this throws an IOException.

If we had more time to work specifically on the question, we would like to confirm our findings above by using a fuzzer to find input (if any) that would generate exceptions in the desired areas, similar to the fuzzing that was done to look for algorithmic vulnerabilities in space for Calculator 1 using the ECJ framework."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.5.2 Question 005 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"A majority of the suspicious methods were located in the Mather class, with many calls being made therein to the BigInteger class (also in the mather directory). As such, these two classes were subjected to thorough analysis. It was discovered that, while there are limitations on the exponentiation method to prevent extremely large expressions from being processed (e.g. 9^9^9), there is no limitation on multiplication. Therefore, we can try to produce very large numbers, e.g. (2^16639)(2^16639)...(2^16639) or (99^2559)(99^2559)...(99^2559). Individually, 2^16639 or 99^2559 expressions take a short but noticeable time to complete. However, by encapsulating the expressions in parentheses and repeating them in a sequence [e.g.

(99^2559)(99^2559)(99^2559)....] an attacker can force the server to compute each of these expressions individually, which can result in a large computation time.

It was also found that in calculateOutcome a queue of tokens obtained from the input expression were processed. Through varying the input it was also found that should there be an unmatched left parenthesis in the initial user input, the queue upon tokenization will "match" that left parenthesis with another left parenthesis. This fact meant that, although the input budget was only 60kB, an attacker can effectively double the size of the expression that must be processed by the server with an input consisting solely of a long string of left parentheses."

Take-Home Evaluation:

The challenge contains an intended vulnerability in the parsing of input expressions. The vulnerability reported by Vanderbilt was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program has and unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.5.3 Question 009 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"Initially we analyzed heavily nested memory allocations with the help of Janalyzer, finding the most nested allocations occurred in part.plain (which was not used in Calculator 2), and in the reformulating methods (invoked by various mathematical operation methods within BigInteger).

Therefore, we have the idea which is to make a number as big as possible with many intermediate smaller numbers involved. Therefore, we try a repeating sequence of (2^16639)'s, to make the final product as large as possible. However, we can only observe about 1.2GB in the process, which is much less than the asked 4GB."

Take-Home Evaluation:

The challenge program contains an unintended vulnerability discovered by Northeastern, Iowa, and Two Six Labs.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"After discussions with Iowa State, we realized there is an exploit. They said their fuzzer revealed that an input of the form:A x B where A is 48000 digitsB is 1001 digitsWill trigger the vulnerability. We verified that this exploit does trigger the vulnerability."

Live-Collaborative Evaluation:

Following collaboration with Iowa, Vanderbilt was convinced by Iowa's identified unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.6.6.2.6 CaseDB

A.6.6.2.6.1 Question 019 (SC Space, Intended Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“The first part of the question asks whether an attacker can discover if the response to a privileged user's GET request contains one or more informants. In fact, since GET retrieves one case, and one case cannot have more than one confidential document, the case is limited to either having zero or one informants. This part of the question, then, boils down to whether a privileged user's GET request contains or does not contain an informant. We used Profit to make a privileged user retrieve (GET) similar cases with and without a confidential informant in them. It detected some leakage, but the initial results were inconclusive because the CaseDB wire protocol interleaves the content in a way that specifically complicates the type of ordered-stream-of-packets analysis that Profit employs. We also used JAnalyzer to track the flow of information and perform taint analysis, and although it helped keep track of some flows, we also found CaseDB to be a very challenging subject for the tool, due to its heavy reliance on the Apache Camel framework and SEDA architecture.

Regarding the exchange of information through the network, CaseDB transmits the documents and their metadata from five nodes, and the documents are distributed over those nodes pseudorandomly based on the document ID, which is set on creation and cannot be modified. This makes it difficult to conduct repeatable experiments without rebooting the server before each trace. Although this prevented Profit from finding the leakage on every single trace, it still found occasional correlations, which pointed us to packets associated with the presence of a secret document. After looking into said packets we detected the pattern "eHh4eHh4" (which is "xxxxxx" encoded in Base64), which is present in the response of a privileged GET if and only if there is a confidential informant's document in the case. This led us to realizing that the packets contain Base64-encoded information, which was hard to decode at first because (i) it is preceded by 8 bytes of non-Base64 content, and (ii) it is not always padded. After separating the first 8 bytes and fixing the padding when needed, we found that the rest of the content can alway be decoded as Base64. Among other things this implies that, when inspecting the traffic generated by a privileged user's GET request, we can tell not only whether it contains a confidential document, but also the unique ID of said document.

However, as to finding out the identity of the confidential informant when one is present, we analyzed the traffic associated with ADD, GET, and SEARCH operations and saw no evidence of a side channel in space that could be powerful enough to obtain that information. The peculiar ordering of packets stems from the way documents are distributed across nodes, which has its origins in the ID of each document, which in turn is assigned by order of creation (which is unrelated to the identity of a confidential informant). The rest of the elements present in the various communication channels and messages exchanged through the network do not suffice to reveal the hidden identity.”

Take-Home Evaluation:

The challenge contains an intended vulnerability that combines leaked information from the target user’s request with a caching behavior and a deterministic algorithm to fetch documents to leak the secret. The team identified the first side channel that leaks whether the response to a request contains an informant but missed the ability to use the caching behavior and

deterministic fetching algorithm and active operations to determine the secret via a character-by-character exploit.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt was unable to identify the intended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.6.6.2.6.2 Question 021 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“The first part of the question asks whether an attacker can discover if the response to a privileged user's GET request contains one or more informants. In fact, since GET retrieves one case, and one case cannot have more than one confidential document, the case is limited to either having zero or one informants. This part of the question, then, boils down to whether a privileged user's GET request contains or does not contain an informant. We used Profit to make a privileged user retrieve (GET) similar cases with and without a confidential informant in them. It detected some leakage, but the initial results were inconclusive because the CaseDB wire protocol interleaves the content in a way that specifically complicates the type of ordered-stream-of-packets analysis that Profit employs. We also used JAnalyzer to track the flow of information and perform taint analysis, and although it helped keep track of some flows, we also found CaseDB to be a very challenging subject for the tool, due to its heavy reliance on the Apache Camel framework and SEDA architecture. Regarding the exchange of information through the network, CaseDB transmits the documents and their metadata from five nodes, and the documents are distributed over those nodes pseudorandomly based on the document ID, which is set on creation and cannot be modified. This makes it difficult to conduct repeatable experiments without rebooting the server before each trace.

Although this prevented Profit from finding the leakage on every single trace, it still found occasional correlations, which pointed us to packets associated with the presence of a secret document. After looking into said packets we detected the pattern "eHh4eHh4" (which is "xxxxxx" encoded in Base64), which is present in the response of a privileged GET if and only if there is a confidential informant's document in the case. This led us to realizing that the packets contain Base64-encoded information, which was hard to decode at first because (i) it is preceded by 8 bytes of non-Base64 content, and (ii) it is not always padded. After separating the first 8 bytes and fixing the padding when needed, we found that the rest of the content can alway be decoded as Base64. Among other things this implies that, when inspecting the traffic generated by a privileged user's GET request, we can tell not only whether it contains a confidential document, but also the unique ID of said document. Regarding the second part of the question: to find out the identity of the confidential informant when one is present, we analyzed the traffic associated with ADD, GET, and SEARCH operations and saw no evidence of a side channel in time that is powerful enough to reveal that information. We timed insertions of items with names lexicographically above and below the secret, as well as with names that shared and that did not share common prefixes with the secret. We timed GET requests with secrets placed in different lexicographical orderings w.r.t. the rest of the documents in the case. We also investigated the

search capabilities, which looked very promising (as they include, for example, the ability to search for documents with partynames lexicographically above and below a given string). But we confirmed that searches are indeed conducted on a database that does not contain confidential documents. We found it hard to conduct unbiased experiments under stable timing conditions. Unbiased experiments ideally require changing one aspect while leaving all others constant (same names, same lengths, same orderings, etc), which is not possible in a context where creation-order IDs dictate behavior, insertions cannot be undone, etc. On the other hand, resetting the server between each attempt biases timings as it makes it impossible to warm up the server. The fact that the order of documents on the wire is mixed up also makes it harder to associate particular observables with particular phenomena, and limits us to coarser-grained observables (e.g., whole-trace duration as opposed to delta between a certain pair of packets). In our investigation, we have not detected a timing side channel that would enable an attacker to identify the identity of the confidential informant."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"The restricted nature of the character set of thesecret suggests that some kind of segment oracle attack might be possible.We collaborated tightly with GrammaTech and Northeastern, the two other teams that worked on CaseDB. We exchanged many leads and ideas, and found several new leads together, but ultimately none of these leads proved to be strong enough to change our answer to a Yes.Northeastern showed that there is a caching mechanism that causes previous query results to be reused. Suppose the privileged user GETs a case with a secret document. This causes an equality SQL query, e.g. "(casename='CASE1X')", to be executed and cached. If an unprivileged user then does a SEARCH with the exact same query string, the query results are read from the cache, and the info lines that are distributed to thenodes contain the secret. The "nodemap" in the server debug output indeed show the value of the secret. The secret is then filtered out by the node before the lines and documents are sent back to the client. This looked very promising, as it shows an internal leak within the server where a secret piece of information makes its way to the node (and is manipulated by more potentially vulnerable code) even though the user doing the search is unprivileged. This was interpreted by Northeastern as a strong indication that a vulnerability was present, although they did not know how to exploit it.GrammaTech showed that the caching mechanism also leaks to the nodes (as described above) when the unprivileged SEARCH is a conjunction, if one of the conjuncts is an exact match of the prior privileged GET query, e.g., "(casename='CASE1X' AND partyname LIKE 'J%')". This looked promising as we thought it could perhaps provide a way for an unprivileged user to perform lexicographic queries over the secret. Moreover, each conjunct is cached separately and contributes all of its matching search results to the nodemap, so that conjuncts that result in non-disjoint sets of documents indeed cause the documents that were matched by multiple conjuncts to appear on the nodemap multiple times. (The duplicates are removed later on, before sending the response to the client.) This also looked promising, as it may provide a mechanism for amplification of a timing difference. GrammaTech also mentioned a limitation that they encountered, where multiple appearances of LIKE clauses would trigger an exception and ruin the experiment. We proposed using SQL "<" and ">" operators instead of LIKE to overcome this, which did not cause such an exception.We also revisited the question of whether the

SEARCH mechanism (which is not supposed to return any secret information, regardless of whether the user is privileged) ever iterates over secret information. When the user is privileged, this seems to happen,There is a mechanism (an alteration introduced by the Red Team in the CQEngine library, which is in scope because the whole library codebase is pasted into the CaseDB codebase) that examines the stack trace during runtime and changes iteration behavior toa less safe version when the SecureDocControllerX class appears in the stack trace, so that secret documents are iterated by queries. We tried to use JAnalyzer to help us establish when this happens, and the

other teams used their own tools, but it was hard to obtain a conclusive result in this way because of the dynamic nature of the condition (crawling the stack trace) which is difficult to overcome for static analysis tools. After some additional manual inspection, we believe this only seems to happen when a privileged user performs a GET request, and when the nodes do certain operations. We could not find an execution path that is triggerable by an unprivileged user where any SecureDocControllerX method is in the stack trace by the time the iteration takes place.However, we were still not completely sure about this. And there is also the possibility that when the caching mechanism is exploited to leak the secret to the nodemap, and additional query conjuncts are added that do and that do not match the secret depending on the comparison provided by the attacker, a timing difference may arise. And we had what seemed to be a potentially viable way to amplify it. So we used Profit to repeatedly run experiments where the caching vulnerability is triggered and a series of additional conjuncts are added which predicate on the secret being above or below a certain string constant. We repeated the additional conjuncts many times (within the 330 character limit for queries). We executed this with the secret set tobe below and above the constant, respectively, with 1000 iterations each. Profit's leakage quantification found no significant leakage in the results of these experiments.We also considered the possibility of a lexicographical attack based on exploitingdelays in the index data structures that are used for searching, which also needs to be maintained on insertions, so that perhaps inserting values below and above the secret might be of use. We did not find any significant timing difference due to this. Our experiments were not exhaustive due to the complexity of the CQEngine library codebase and our limited knowledge of its inner workings."

Live-Collaborative Evaluation:

Vanderbilt correctly kept their No answer after collaborations.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.6.3 Question 036 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"After familiarizing ourselves with the decompiled code we determined that class GetCase in com.ainfosec.casedbcamel.svr.queryhandler manipulates the secret information (both key and password). We applied SPF-SC for the side channel analysis of method send() in GetCase. The analysis revealed a side channel due to extra operations (throwing exception and catching it) during the byte-wise matching in the "for" loop. […] The alphabet has 26 + 10 + 3 = 39 chars.

Since the secret password is at most 8 chars long, it can only involve up to 8 different chars (or less, if there are repetitions). So, we know that at least 31 chars of the alphabet are unused.

1. For each of the 39 chars in the alphabet, try a guess consisting of only that character repeated N times (e.g. "aaaaaaaa", "bbbbbbbb", etc). So in 39 tries, you find out which chars are unused, which chars are present, and how many times each of them is present.
2. For each of the at most 8 chars that are present, try 8 guesses consisting of that character in each possible position, and some unused character in all the rest. For instance, if you know that a is present but z is not, try: "azzzzzzz", "zazzzzzz", "zzazzzzz", ... "zzzzzzza". So in 8 tries you find out which of the 8 positions are 'a'. In total you need 8x8 = 64 guesses.

So, in 39 + 64 guesses you should get the password, which is within the budget. However, it is not clear if this side channel is observable on the server side, although the "throw catch" code might be enough to make it observable. Furthermore, the attacker would also need to uncover the secret "key" but we could not determine a side channel that could recover it. We analyzed the encryption/decryption used by the application, but we could not find anything (except that the iv codes provided by a malicious client may not be properly randomized). We therefore conclude that the likely answer is "No".”

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“We collaborated tightly with GrammaTech and Northeastern, the two other teams that worked on CaseDB. We exchanged many ideas, and devised a potential attack that would work, even if we don’t have the benign user’s key, as long as the side channel that reveals how many characters are mismatched during the password comparison is strong enough,. However, after additional experiments, the latter still seems to not be strong enough.During collaboration we noticed that the configure() method of ServerRouteBuilder explicitly connects the CASEDBLoginException (which is thrown for each character mismatch during comparison of the unencrypted passwords) with a logging mechanism, and seems to use a synchronous call and wait for the task to complete:this.onException(CASEDBLoginException.class).handled(false).to("seda:log?waitForTaskToComplete=Always").to("netty4:tcp://127.0.0.1:5156?sync=true");We saw this as suspicious, combined with the unnecessary throwing of such exceptions, and this renewed our expectations that perhaps we could find a way to infer the number of mismatches from a timing side channel, or perhaps a way to turn on a more expensive form of logging that would amplify the side channel.Our new idea is based on the fact that initial vectors for the AES encryption are provided by the client, that the password is short enough (5 to 8 characters) that a single block is used by AES. Thus, the user-supplied IV is simply XORed once with the result of the AES decryption of the unknown password usingthe unknown key.Thus, without knowing the password or the key, a modified client could send carefully chosen IV values (without any repetitions, as this is forbidden by the server) in order to progressively reduce the number of mismatches until reachingzero mismatches. This can be done within budget.We used both Profit and manual inspection to examine the packets involved in unsuccessful authentications, using wrong passwords that have {1, 2, 3, 4, 5} mismatches. We found no significant leakage. The number of packets exchanged during unsuccessful authentications is very small, so the number of features is quite small. Therefore we are rather convinced that there is no leakage there. And

we did not find a way for the attacker to turn on additional logging, either using JAnalyzer or manual inspection.

We also used Profit to simulate the benign (privileged) user's successfulauthentication, by changing his key and password (on both the client and the server) to different values, executing GET requests repeatedly (that includes successful authentication) with each different value, and looking for leakage from the timings and time-deltas of all packets involved. Profit's leakage quantification did not find significant leakage here, either."

Live-Collaborative Evaluation:

Vanderbilt correctly kept their No answer after collaboration.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.7 Chessmaster

A.6.6.2.7.1 Question 006 (SC Time, Intended Not Vulnerable, Take-Home: Yes, Live: No)

Take-Home Response:

"Then, we used Janalyzer to get an initial understanding of the problem. We found that there is a method called "nextMove" that tries to evaluate state of the game and decide which move to take next. The algorithm is deterministic, and depending on the state of the game, it always decides on the (local) best move from all the possible moves. Moves are compared using the scores computed based on the type and count of pieces in the game. The algorithm tries to predict the player's next moves and it generates a tree, where each node is a possible next move with a score.

Applying DifFuzz, our new tool for automatically identifying side-channels, showed that indeed there is a side-channel in the nextMove method where the state of the game is the secret. DifFuzz is a fuzzing-based approach which automatically detects side-channels in time and space by analyzing two versions of the program to find inputs that maximize the difference in resource consumption between secret-dependent paths. To run DifFuzz, we need a driver that parses the inputs from the fuzzer and executes the two copies of the code under test, while also measuring the cost difference. For this problem, we wrote a driver that reads the state of the game for two games and passes each of them to the "nextMove" method. The driver also validates these inputs and ignores those that are not valid states in chess. For example, it verifies if there are still two kings in the game, or when a pawn has reached to the last row, then it promotes it to a queen. You can find the driver attached. As part of the DifFuzz, AFL generates input files, which get parsed by the driver. The application is instrumented and the fuzzer runs the instrumented application with random inputs and attempts to maximize the user-defined function, which computes δ, the cost difference between two executions. The result of the analysis is a set of concrete secret inputs that expose the maximum cost difference between two secret-dependent paths found by the fuzzer. If the difference is large, that indicates a side-channel vulnerability. Interestingly, only after 3 minutes, DifFuz could reach a large cost difference between two executions (δ= 4,046,687,707). This shows that depending on the input, two executions can have more than 4 billions of byte code instructions difference. This obviously indicates a side-channel.

Looking more closely into the code, we found that depending on the state of the game, the number of nodes in the decision tree as well as its depth are different. For example, depending on the state of the game and how strong the other player is, the nextMove method computes a depth, to predict the second, third, or even the eleventh next move of the player. The bigger the depth is

more computations are needed to determine the next move. Also, if a queen is in the middle of the board it creates more possible moves for the next move, and tree become much bigger.

Exploit and budget:

In chess after first move 400 positions and after fourth move more than 288 billion positions can be created. Thus, the possible states of the games is obviously above the budget. However, in this application the AI move (black side) is deterministic based on the state and also there is a time side channel on the nextMove method. We show that having this information it is possible to decrease the number of possible states drastically to be in the budget. Then, using the oracle it is possible to check if the predicted state(s) are correct. Note that in this application the white's moves (target's moves) are secret. In the first state, in chess always, the player (here white) has 20 possible moves. Thus, the attacker on different games tries all the 20 possible moves one by one and stores the time needed by the AI to compute the nextMoves. Then, by comparing these obtained times with the actual time captured during the target's game, the attacker can predict the move (or a subset of moves) among these 20. Then, for predicting the second move the attacker does the same steps to minimize the possible moves. This will prune the tree. In practice, a chess game is finished in 15 to 80 moves, while the average number of moves in a game is 35 moves [1]. Also, in each state the possible number of moves for white are limited, on average about 40 moves and in rare cases about 120 moves. This number of moves are decreased drastically by using the information obtained from the side-channel in time, e.g., depending on a time similarity threshold, we found many states with 40 possible moves were decreased to 1-10 possible moves. We implemented this attack and ran it on a few examples. In particular, in our experiments, we found that for the first move the attacker is able to predict one to four states among 400 states. Then, for the second move it could predict among 10 to 30 states out of 72,084 states. Also, for third move it could predict around 90 states among more than 9 million states. This indicates how fast the tree is pruned. Here, we provide the attack that uses this vulnerability.

Note that we assume that the benign user is honest and do not maliciously put the game in an infinite loop since we found that this application has a bug that a game can be continue forever. For example, it is possible that white player moves its knight from g1 to f3, which in return the black player (AI) moves its rook from a8 to b8. Then, if white player moves back knight from f3 to g1, in return black player moves back from b8 to a8, and this loop can be played forever without the game being finished. Note that this is against FIDE's law (world chess federation) where a game is announced draw if a position is repeated three times. Considering this malicious game then the predictions cannot be in the budget. In other words, this attack is in the budget by this assumption that the benign user is not going to put the game in an infinite loop and he/she plays the game honestly."

Take-Home Evaluation:

Vanderbilt proposes an exploit that appears to violate one of the question's assumptions – that the target user is in the process of playing a game. If the attacker is able to create a new game, then the target user is no longer in the process of playing the game in question. Additionally, it is unclear how the attacker can change the game state without this information. Without the data to back up the potential time side channel in the computational time for a move the be made given knowledge of the deterministic AI algorithm, we don't believe this exploit is feasible.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Before starting the live engagement an assumption was added by EL that the game can be continue up to 10,000 of moves. This assumption means that loop in the game is possible. It is mentioned in the last paragraph of our original take home “Analysis” section that our attack will be out of the budget if loop things are possible. Actually, ten thousands of moves is considered as an infinite moves for a real chess game. Based on the [1], in usual chess games average of moves are 35 and a very low percentage of games can go more than 100 moves.

Thus, by the assumption of ten thousand moves our attack cannot guarantee to be in the budget and our answer is changed to NO. However, it should be noticed in our original take home it is mentioned that with looping (like 10,000 moves) assumption our answer is NO.”

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge is not vulnerable. Note: without the provided assumption of a maximum 10,000 moves by the target user, the challenge was trivially non-vulnerable as the worst-case game would exceed the available budget for observation.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.7.2 Question 015 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“Then, we used Janalyzer to get an initial understanding of the problem. We found that there is a method called "nextMove" that tries to evaluate state of the game and decide which move to take next. The algorithm is deterministic, and depending on the state of the game, it always decides on the (local) best move from all the possible moves. Moves are compared using the scores computed based on the type and count of pieces in the game. The algorithm tries to predict the player's next moves and it generates a tree, where each node is a possible next move with a score and its children are the next possible moves after this move.

We ran KelinciWCA to find states of the game that trigger worst-case in the application and nextMove method. KelinciWCA extends Kelinci with prioritization of costly paths. Costs are collected on the Java side, then sent to an extended version of AFL. The fuzzer maintains a current highscore with respect to the used cost model. When creating the next generation of inputs, the fuzzer selects ancestors from the set of inputs from the previous generation that either lead to the execution of previously unexplored program branches or to a new highscore. The chance of an input being selected from this set depends on its score, as recorded on the Java side.

We used the user-defined costs functionality of KelinciWCA. This functionality allows a relationship between input and cost about which a machine can reason. We implemented a driver that reads and parses the inputs generated by the fuzzer and pass them to the chessMaster application while also measuring the number of jumps as the cost. For this problem the inputs are states of the game. The driver also validates these inputs and ignores those that are not valid in chess. For example, it verifies if there is still two kings in the game, or if a pawn has reached to the last row, then it promotes it to a queen. […] if the attacker logs in with three different username and passwords with some delays between them and plays the above game, then it causes the real runtime of the benign request to exceed 1000 s. Note that even though the games include 3*14= 42 moves, however the benign request exceeds 1000 s when one game is at its last stages of the game, the other two are in middle stages of the game. Thus, overall the attack is in the budget. When exploring the problem, also we also came up with the following attacks, however none were in the budget:

Attack 2:

It is possible that the attacker provides an state of the game that makes the game to continue for ever. The attacker just moves one knight piece back and worth and the chessMaster in response only moves Rook piece back and worth for ever. Also, the attacker with the same username and password can start several parallel games on chessMaster, and if he runs the game that goes on forever, then the sever cannot response to the request of a benign user's move. However, since this attack involve many moves, then it would not be in the budget.

Attack 3:

Also, there is a promotion method in the application, which is called when a pawn reaches to the final row. In this case, the pawn can be convert to a queen which creates many possible moves for the next move. Moreover, the value of depth in the decision tree depends on how well the players are playing. At the beginning when both players have the same score (e.g., having the same number of pieces in the game) then the depth is set to 3. But, when AI loses some pieces it increases the depth so it can have a better prediction for the next move. We observed such scenario using KelinciWCA where for example, in a game the AI has just one king while the attacker has several queens (i.e., some of his pawns are converted to queen). However, reaching to such state of the game requires more moves (requests) which is higher than the budget.

Attack 4:

As mentioned sooner, attacker can start parallel games by the same username. Also, there is a race condition problem in the server “this.client.state.move(aiMove, ' ');” that different games can modify the state (and board) simultaneously. Thus, unpredictable states can be created when the attacker run different games at the same time. In this scenario the state can reach to a position which generating the next move takes a long time. However, because of its randomness it is not always successful and also it might not be in the budget.”

Take-Home Evaluation:

The team hinted at the intended vulnerability in pawn promotion but did not appear to note that this vulnerability can be used to get 2 kings within the input budget and exceed the resource usage limit. Instead the team noted a vulnerability in reaching a computationally expensive state with parallel requests to delay the benign request. Vanderbilt’s response appears to be an unintended vulnerability that was confirmed.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

“As it is mentioned in our original take home “Analysis” section, the Kelinci (our fuzzing tool) found if we have two kings with the same color in a board (which is illegal in a real game and in this application) and the other color has one King the AI cannot make a decision. Because in the program the weight of the King is much higher than other pieces. Thus, in this scenario the integer “depth” value which is used for generating the tree for finding the best move will be much bigger than usual. Thus,the depth of tree and analysis by AI become much more bigger and time of decision in the “nextMove” method can go for some hours. The Colorado Boulder University found that when they are promoting a pawn they can write a typo for King like “Kng”. It will be checked in the “promotion” method that the pawn is not going to be promoted as a King. In this case it is not, and “Kng” is not consider as a “King”. Then, the “Kng” will be

compared with all of the piece's names. As, the "Kng" in none of them the "autoCheck" method will be called and it will change the name of "Kng" to "King" and without any other checking it will be promoted to "King". However, without using two kings in one side we could reach several positions by Kelinci that the benign user cannot access the server for more than ten minutes, and also we created an attack based on a real game to discover a real game which can led to one of these positions, and a sample game is in a take home example. In conclusion, the answer is Yes."

Live-Collaborative Evaluation:

After collaboration Vanderbilt reported the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.8 Class Scheduler

A.6.6.2.8.1 Question 012 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"First, we played with the application using the scripts provided with the problem. The user can upload XML files to the application. We applied DifFuzz, our new tool for automatically identifying side-channels on the upload method of the application. This analysis showed that indeed there is a side-channel in the upload method where the number of teachers and students are secret. DifFuzz is a fuzzing-based approach which automatically detects side-channels in time and space by analyzing two versions of the program to find inputs that maximize the difference in resource consumption between secret-dependent paths.

To run DifFuzz, we need a driver that parses the inputs from the fuzzer and executes the two copies of the code under test, while also measuring the cost difference. For this problem, we wrote a driver that reads the number of teachers, students, courses, and classrooms. Then, using these numbers it generates XML files that can uploaded to the server. The driver also validates that these numbers are in the range acceptable by the application. […] As part of the DifFuzz, AFL generates input files, which get parsed by the driver. The application is instrumented and the fuzzer runs the instrumented application with random inputs and attempts to maximize the user-defined function, which computes δ, the cost difference between two executions. The result of the analysis is a set of concrete secret inputs that expose the maximum cost difference between two secret-dependent paths found by the fuzzer. If the difference is large, it indicates a side-channel vulnerability. Interestingly, in less than 2 minutes, DifFuz could reach a large cost difference between two executions (δ= 15,603). This shows that depending on the input, two executions can have more than 15,603 of byte code instructions difference. This obviously indicates a side-channel.

Looking more closely to the code, we found that during the upload, the information in the XML file is extracted using "for loops" that depend on the secret values (number of teachers and students). This dependency causes the side-channel. Also, this side-channel is amplified because in the upload method, before extracting the information from the XML file, the XML file is getting validated, which also needs extracting the information from the XML. This explains the high cost difference for two different secret values.

To exploit this side-channel vulnerability, the attacker needs to observe the network traffic and use the correlation between the secrets and the timing needed for uploading them. Offline, the

attacker can upload different XM (sic) files with different number of teachers and students and capture the timing. We used Profit to capture these timing values and tries to see if the side-channel is observable over the network and if it can be exploited. Using Profit, we have done an experiment with 10 courses, 11 classrooms and 12 teachers were set up and the number of students were between 10 and 122 to see if the change in number of students would influence the timing side channel over the network. Our experiments and analysis of network trace showed that 1.2 bits of 6.8 bits (log113) leaked with total running time feature and timing delta between packets 19,20 from clientNuc to serverNuc feature. The plot of distribution of this feature is attached where y axis is probability of secret given observable, x axis is observable, different curves represent different secrets and each sample is distributed below the curve, scattered in y axis to be able to show how it is in general. This means that 5.6 bits of information are unknown and have to be guessed. To guess 5.6 bits of info, we need to make $2^{5.6} = 49$ guesses which is out of budget of 39 oracle queries. We could do more extensive analysis to see if number of teachers would leak but the question asks number of teachers and students both, so if one doesn't leak, we wouldn't be able to get the total secret in worst case. If there were another feature with a different alignment of network traces for this experiment, we couldn't find it and our findings suggest that there's no leakage within budget."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Iowa and Utah had the idea of measuring the percentage bar progress of schedule generation to update timing to correlate it with the secret. They also had another idea of using decimal of percentage value to determine the number of total generations to see if any of those information correlates with the secret. The percentage is updated every 100 + (0, 250) ms in the class VaadinUI$Processor. Schedule generation uses genetic algorithm to generate schedules and number of genes in a chromosome do not depend on number of students and teachers, so it does not seem to affect the computation in the loops. Iowa did some data analysis and guessing number of total generations with percentage worked 85% of the time but it didn't correlate with the secret. They also analyzed the progression of different secrets that are run multiple times with linear regression (making it a function that takes the percentage and returns time) to see if the functions were stable and unique for each secret. This didn't work as even for same secret the function slopes vary a lot, creating a lot of collisions. Therefore this question stays No. Both Profit's modeling and this linear regression modeling provided equal approaches, Profit has the advantage of providing leakage calculation but it can't interpret the sequences of values which another statistical modeling helped."

Live-Collaborative Evaluation:

Vanderbilt kept their correct No answer after collaborative analysis.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.8.2 Question 017 (AC Time, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

“First, we played with the application using the scripts provided with the problem. The user can upload XML files to the application, and then request the server to generate a schedule. We applied KelinciWCA on the createSchedule method in the application to automatically identify if there is any algorithmic complexity vulnerability in time in the application. To run KelinciWCA, we need a driver that parses the inputs from the fuzzer and executes the code under the test, while also measuring the costs. For this problem, we wrote a driver that reads the number of teachers, students, courses, and classrooms, as well as number of generations, max students per course and number of sections per course. Then, using these numbers it generates XML files that are uploaded to the server and finally the createSchedule is executed. The driver also validates that these numbers are in the range acceptable by the application. You can find the driver attached. Interestingly, initially we had not put any validation on the number of sections and with that KelinciWCA could found inputs that generating the schedule for them took even more than the budget 50 minutes. However, adding this validation in the driver, after running KelinciWCA for 18 hours, it couldn't find an input that exceeds the usage limit of 50 minutes. However, with KelinciWCA, we can also have parallel experiments that learn from each other. We used our cloud infrastructure with 128 cluster nodes and ran 256 experiments where all tried to increase the cost (here the number of instructions in byte code). After running them for 8 hours, it couldn't find an input that causes the real runtime of the benign request to exceed 50 minutes.”

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Colorado presented an attack which set the section sizes of each class to large negative and positive values and after the file is uploaded, the generation of schedules takes a large amount of time. Because the condition checks only sum of section sizes, and there are loops dependent on section sizes, this hangs the session. Initially, they hanged their own instance, but another instance could try to issue a download request and get “We’re busy, come back later.” reply. Apogee clarified that that counts as a reply to the request. Colorado did the same attack on multipleinstances where one file is uploaded to the server and launched four instances that try to generate the attack. This maxed CPU usage in their local machine and curl requests or requests from another browser fail to launch the website, indicating that the thread pool is all allocated for those generations. This isn’t done on the reference platform but on a local machine on web browsers as it’s extremely difficult to script the interactions with VaadinUI interface. We believe this demonstrates the attack and we are changing our answer to Yes.”

Live-Collaborative Evaluation:

Following collaboration, Vanderbilt reported the same unintended vulnerability as their collaborators.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.8.3 Question 028 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“As it is mentioned in the description, the attacker does not have the ability to passively observe the traffic. The only possible way to learn about a teacher's schedule is by-passing the login.

Thus, we used DifFuzz on the Login operation in the application. DifFuzz is a fuzzing-based approach which automatically detects side-channels in time and space by analyzing two versions of the program to find inputs that maximize the difference in resource consumption between secret-dependent paths. We found that indeed there is a side-channel vulnerability in time for the Login operation. Looking more closely to the code, we identified the Authentication method which uses the java's String.equals() function which is known for its side-channel vulnerability. However, since the length of password can be up to 1024 characters then the search space is as big as 1024*256 which is much bigger than the budget."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.9 Effects Hero

A.6.6.2.9.1 Question 010 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"For this question, one possible attack approach would be to intercept the actual bytes of the output file as they are downloaded. Alternatively, we could intercept the bytes of the input file as they are uploaded, and then somehow extract the names and parameters of the transformations applied to it, in order to recreate the output file on our own replica of the system.

However, the input (original) audio file is uploaded via https, and the output (processed) audio file is also downloaded via https. Their contents are encrypted while on the wire.

We do not believe that any side channel based on the timing metadata of the encrypted network packets can be strong enough to reconstruct the whole contents of said packets. It is just not enough information. Thus, the attack, if any, should consist of finding the location visited, or command issued, by the benign user in order to download the file, and mimic it to re-download the file. (This requires some sort of caching to take place on the server side, i.e., the output WAV file must not disappear immediately as a consequence of the benign user obtaining it.)

The Vaadin framework works by sending JSON snippets to the app server which correspond to interactions with the remote interface, such as a mouse click on a widget like a button. We considered mimicking the benign user's action of clicking on the "Download .wav" button. In other words, finding a way for the attacker to emulate a second click on the same button.

Emulating a click on an interface widget (such as the "Download .wav" and/or the "Run" buttons) of someone else's UI requires knowledge of their uiID. The uiID is a short number, and could indeed be leaked somewhere. But it also requires knowledge of the CSRF UUID, a full-length unique identifier that is used to deter cross-site request forgery, and which is handled

automatically by the Vaadin platform. There is no mention of it in the codebase within scope, so the code within scope never manipulates that secret.

However, it turns out that re-clicking the download button neither suffices nor is necessary to re-obtain the file. It is not sufficient because clicking on the button merely instructs the client to make another separate https request to a different location. And it is not necessary because, after the "Download .wav" button is clicked by the benign user and he or she downloads the file, the same download location remains active for a long time and can be reused at will.

Hence, in order to re-download the file, two things are needed:

1. The attacker must know the location where it should be downloaded from. This location is always of the form https://<serverHost>:8443/vaadinServlet/APP/connector/<uiID>/<resID>/dl/Output.wav, but the attacker must find out the uiID and the resource identifier (which appears to be the ID of the download button, plus one).
2. The attacker must also know the session ID from the benign user's cookie. This is a standard cookie with a 256-bit (32-char hex) session ID, for example B0EAF147F80256F6170F4777D9781065.

If the attacker knows this information, he/she can easily re-download the file from any computer, and the server will honor the request. But the cookie (i.e., the session ID) is necessary. Without it, the server will deny the download request. All session management, including cookies, is handled by the underlying frameworks automatically. There is nothing in the codebase in scope that deals with session management or accesses any cookie-related variables. Therefore, we consider that any vulnerability that may reveal the session ID is out of scope."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.9.2 Question 025 (AC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"To delay the session request, we thought about launching the program and locking the state for a while so that it would delay to respond to the benign request. We observed that the runStage() function in VaadinUI.java had a loop with the parameter Duration and FileInputBlock had 2 loops in the upload, dependent on the sample rate, number of channels in the sound file and the length of the file. We used KelinciWCA to fuzz some example .wav files to obtain .wav files that would run for more executions but it didn't generate anything that runs for more execution time. We think that while fuzzers can generate valid wav files by flipping bits, it's difficult for a fuzzer to randomly discover that the sample rate or more channels would increase the execution. Also those parameters the next parts of the file as well, more channels require more bits to represent

content in those channels. We used SoX to generate 1 second .wav files with 10 channels and 2 times the usual sample rate of 44100 Hz which had 3.6 MB size. When we uploaded it using FileInputBlock, we noticed that it was running in background and it didn't affect the initialization or the stage running in any shape or form, so we couldn't amplify it. We also tried initializing session while another session is running with 50 FunctionGeneratorBlocks and 30 seconds of duration. While the server takes 7 seconds to process this information, initialization wasn't affected and the running time wasn't altered. We also played with the initialization parameters in the initial cookie like window size, screen size, time zone delay, etc. to see if that would slow down the execution but that didn't work either. Each new initialization took a new UI Id, so we tried to set the UI Id using the initial cookie but that didn't work as the server set the UI Id each time with new sessions and without a delay. We didn't examine the VaadinUI library extensively but empirically, we couldn't discover a way to delay the benign request."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Iowa had an attack using FileInputBlock with a wav file with 32k channels and one frame for each channel and using that block for all 50 blocks and setting the sample duration to 30 seconds. The processing takes more than 450 seconds and total upload is 1638 KB which is within budget. Colorado, Grammatech used 2 FunctionGeneratorBlocks and 48 BeatBlocks with BeatMatrix BPM set to a high number (like 400000) to invoke the delay instead. Utah used BeatBlocks in the same fashion as well but changed the order of operations so that each block depends on the next, and that delays it for an extra amount.We tried to replicate Grammatech, Iowa, Utah and Colorado exploits because we observed non-blocking behavour in our experiments. They all block the behaviour when we try to launch the website on the same browser in a different tab but when we try to launch the website on another browser when the exploit is running, it successfully initializes the session in at most 5 seconds. Also the bash script that the question asks us to delay works pretty fast and successfully as well, we get a reply to our request in time. The reason seems to be that VaadinUI allocates threads to each user and it may check that the same user is trying to access it and block that thread for it.Colorado investigated the budget with 10 BaseBlocks which would block it more than 7.5 minutes but it used ~250KBs request size per session. Assuming the limit of threads is 20 (Budget/Request size), we can block each thread but we also tried initializing more than 200 sessions with curl and it worked each time, showing that thread pool is not that small. We believe the answer is No as we can't block the system and delay them from serving other users in any way."

Live-Collaborative Evaluation:

Vanderbilt kept their correct No answer after trying to replicate potential exploits other teams found.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.10 Railyard

A.6.6.2.10.1 Question 013 (AC Space, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

“The Janalyzer identifies one allocation that takes place in a recursive call (in the com/ainfosec/Util/checkDirectory method). This method recursively looks for Car classes. This recursion depth can be increased by, for instance, adding a large number of recursive symbolic directory links to the initial directory that is searched. However, we believe this is out of scope for an attacker. Our next idea was to monitor the memory consumed by the file complexity attack we identified in Q018. The Janalyzer shows one new allocation inside this infinite loop, which is the CargoItem.toString() method. However, the garbage collector seems to reclaim this memory before it can get close to the budget. The Janalyzer reports that the remaining "new" allocations take place inside only one-level deep loops in the Pack, linkCars, buildMaterials, packCars, and the printToStream methods. However, these all appeared bounded, too, and we were not able to trigger the memory limit.”

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Utah initially presented a potential attack that far exceed the budget. The group consensus was that this attack seemed infeasible. Colorado said they experimented with using the infinite loop attack from Railyard Question 018 and then trying to “break” the infinite loop so that the code would then proceed from writing to the file in an infinite loop (using the fileStream) to writing to a StringStream (in memory) in an infinite loop. However, they said they were unable to trigger this. Our team brought up the server and began experimenting with this idea since it seemed plausible that, if the timing of the requests was correct, a second set of requests could break the infinite loop (by setting the static “next” field of the CoalCar class so that there wasn’t an infinite loop) and then quickly resetting the static “next” field of the CoalCar class to re-establish an infinite loop. We weren’t able to do this using one platform, then the group realized there are multiple platforms, and so we began experimenting with using multiple platforms (since the static “next” field of the CoalCar class is the same regardless of the platform). We were trying our experiments on a local laptop, which wasn’t configured with the same heap limits as the original program, so we began experimenting on the NUC. After trying the above attack on the NUC, our exploit was able to exceed the resource usage limit, but not every time due to the following race condition: Server thread 1: print platform A to file in infinite loop server thread 2: link cars in platform B such that the infinite loop in thread 1 is broken race condition: re-link the coal car in platform B such that the infinite loop is re-established after lunch, Utah said they re-tried the attack but added 100 cargo items (of quantity 0) to platform B before sending out the train on platform B, which then triggered the vulnerability consistently with memory usage always exceeding the resource usage limit in all of our trials. Thus, we changed our answer to yes, there is a vulnerability present. To trigger the exploit:

1. Run the attached script “bad_a.sh” 2. After bad_a.sh has entered the infinite loop writing to the output file, run the attached script “bad_random.sh”. This script breaks the infinite loop and then reconnects it when the “bad_a.sh” script is writing to memory. This was a very good collaboration where initially, every team thought there was either not a vulnerability or that the vulnerability far exceeded the input budget, and in a very short time, and working together, we were able to combine our ideas into a working exploit”

Live-Collaborative Evaluation:

Following collaboration, Vanderbilt reported the same unintended vulnerability as their fellow collaborators. The team noted that this identified vulnerability came out of a joint collaborative effort where no single team discovered the vulnerability during the take-home engagement.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.10.2 Question 018 (AC Space, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“We first identified potential files and the locations where they are written. The printToStream method in the Platform class is easily detected as the location where most of the writing to the log file occurs. Janalyzer detects a loop inside this same method as having a "suspicious" exit condition, namely that the file is written to as long as the current object is not null. Manual inspection of this code reveals that the current object is set using reflection (which, as a side note, is difficult for our tools to reason about). Inspecting this reflection shows that it is the "next" field of the current car that is being retrieved. Quickly inspecting this "next" field of the different types of cars reveals that for a coal car, the next field is static (the "next" field is an instance variable in all other cars). The question then becomes, "can a static next field somehow create a cycle or loop through which the current object in the printToStream method is never null?" We then looked to identify where the "next" field is set and identified the linkCars() method in the Platform class. This method also uses reflection to retrieve and set the field. We saw that the "next" field is not set for the last car in a train. Combining this with our knowledge that the "next" field of a coal car is static, then the question became, "can we either have two coal cars in the same train, one of them being the last train, or can we link trains between platforms?"

The AddCar method in the Platform class revealed the final piece of information needed: a case-sensitive String comparison allows two coal cars on one train. Once these two coal cars are inserted, the suspicious loop in the printToStream method can loop forever and potentially exceed the budget for the file size. We confirmed this with the included exploit script (bad.sh).”

Take-Home Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt identified the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.10.3 Question 035 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

“We first checked what information the active operation could provide to the attacker. Because the attacker doesn't have the password, they appear unable to trigger any operations on the server that are affected by the cargo materials (they are rejected because they aren't authenticated). Next, we used Wireshark to passively observe the sizes of the encrypted Application Data packets generated using the included scripts that cause (1) only a single cargo item to be

contained on the schedule, and (2) all combinations of two cargo items to be contained on the schedule. For both the single item and two item case, we were able to cause the sizes of encrypted packets to be the same by varying the length of the "Description" of the cargo item. Thus, we concluded there is no side-channel available to an attacker."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.11 SimpleVote 3

A.6.6.2.11.1 Question 001 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Each ballot is associated with a registration key. In order to view/change/submit a ballot, the corresponding registration key must be provided. This is done in /registrationkey webpage upon clicking on an election. The code which handles this logic resides in the class easydecision.manager.RegistrationCheckManager.java, mainly in the handleParcel() class.

For a registration key to be valid, it must conform to a complex specification. This is done through the check() method in the easydecision.RegistrationGate.java class. There are several validation checks:

The registration key must be of length 12

The first 6 characters must be digits

The last 6 characters can be digits (other than 3,6,7), but may contain the characters 'A', 'C', or 'E', which in turn get translated into 7,6,3, respectively.

The key is then split into two parts: kStr is the first 6 characters, nStr is the last 6 characters. Cast into an Int, nStr must be divisible by 7 and in the range [0, REGISTRATIONGATE.MAX_N]; REGISTRATIONGATE.MAX_N is initally 995, but increases by one each time nStr is larger than it. The last phase of validation is the most complex, occuring in the method RegistrationGate.checkCore(): […] The matches() and logisticMapping() methods perform the last validation check. The matches() method simply checks for equality between the first 6 significant digits of val (which is a double) returned by logisticMapping and str (corresponding to kStr from above).

The logisticMapping() method is far more complex and resides in the mathematic.LogisticMapping.java class. It essentially computes a recurrence relation for the value of integer value of nStr: F(i) = 4*F(i)*(1-F(i)), F(0) = 0.375. Intermediate values for this recurrence relation are stored for quicker computation. Due to the for-loop in the checkCore()

function, logisticMapping() is called for each value 0 <= i <= nStr, whichs leak timing information regarding the value of nStr. Assuming an attacker could retrieve the registration key from the victim voter, another check must be passed. Each voter is given a unique ID associated with their profile, and each registration key is tied to this ID. Once the registration key is checked for validity in RegistrationGate.check(), the application cross references the current user's ID and registration key with known ID/key pairs for that particular election. This is done in stac.easydecisionEasyDecisionServiceImplementation.checkRegistrationCode(): […] Thus, it is not sufficient to simply get the registration key. Either an attacker must somehow change their identity, or add their ID/registration key pair into the application's database. Given the actions an attacker can make as a voter, this does not seem to be possible."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as the Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.11.2 Question 033 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Access to a voter's profile is done through a GET request to /profile which can only be done after a successful login from the /login webpage, which requires a valid username/password combination. The logic for this lies in LoginManager.handleParcel(), which in turn uses TokenedPasswordChecker.processPassword() to verify the login credentials. Once the voter has logged in, a random UUID is assigned to the voter and returned as a cookie, which lets the voter remain signed in for the rest of the session.

The processPassword() verifies the voter's login credentials by building up a token string, while simultaneously checking the given password against the stored password (for said voter). Valid passwords are somewhat restricted, in that they must between 5 and 15 characters in length (inclusive), contain no spaces, and must be characters in the ascii range [! , ~]. Regardless of how many characters are matched, the validation takes an equal amount of time - no timing information is leaked. After validation, the token string is compressed and randomized, and, in the case of an unsuccessful login attempt, is returned back to the user. Again, there appears to be no timing information leaked from the compression or creation of the token string.

Aside from knowing a voter's password, an attacker could view the target voter's profile by either obtaining the random session UUID (obtained by the voter after login), or somehow obtain the voter's unique ID and convince the server that he/she is indeed said voter. Neither of these approaches seem viable through the operations allowed by an attacker."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.11.3 Question 034 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"The question asks if there is a side channel in space in which an attacker can determine the percentage of votes a candidate received from their best precinct (within a certain threshold). In terms of active actions an attacker (acting as a voter) can take, none of them, aside from casting a vote, gives any information regarding the number of votes any particular candidate received. However, each simplevote server (or precinct) periodically sends updates to other servers regarding their vote count. This aggregation process occurs after one of the three following conditions hold:

1. an administrator initiates the update
2. a ballot is updated ten times
3. every 60 minutes the update is scheduled

Since an attacker does not have admin privileges and it is unlikely for the attack to go on for multiple hours, the best avenue for an attacker to glean any information would be to modify a ballot in a way which leaks voting information, and update said ballot ten times in order to trigger an update.

The logic for updating ballots resides in EditBallotManager.handleParcel(). A ballot has a number of questions, some of which are multiple choice with the rest being free response. Of particular interest is a snippet of code which parses the free response and legitimizes the choice […] That is, if the free response starts with whitespace, a handmade trim() method is invoked […] When trim is invoked, ResultsMessageCreator.update1() is called before the trim, and ResultsMessageCreator.updatel() is called after the trim […] These two methods work together to define a the INCLUSION_THRESHOLD which determines which ballots should be included in the election results message. Once a ballot is updated ten times, AggregationService.gatherResults() is invoked, which in turn calls ResultsMessageCreator.buildMessage() to build the results message. During this process, responses are parsed and legitimized further, only being included if they are completely necessary […] Once the message is built, the message is compressed by first building a Trie and using an unusual form of LZW compression on said Trie.

We used the package analysis portion of our tool Profit to analyze what would happen if an attacker attempted to craft responses which manipulated what was included in the election results message. Specifically, we in one of the free response answers, we put the specified candidate with two spaces before the name, and a number of spaces after the name, in order to see if there was any change in the compression scheme which would affect the resulting size of the election message. The number of spaces increased by 5 for each interaction, and we tried both valid and

invalid candidates […] While there is some size information, it is identical regardless of whether the candidate was a valid or invalid candidate. This does not seem to be a side channel."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Colorado, Grammatech, Iowa, and Utah found additional justification for a negative answer. The messages containing the vote results are created in ResultsMessageCreator and use the field INCLUSION_THRESHOLD to determine if a candidate should be included inthe voting results. Since INCLUSION_THRESHOLD is attacker controlled, this could let an attacker control what is included in the voting results. The method ResultsMessageCreator.shouldInclude() compares a value between 0 and 1 to INCLUSION_THRESHOLD to determine what should be included in the message. However, INCLUSION_THRESHOLD is an integer value, so an attacker cannot use this to manipulate the voting results message."

Live-Collaborative Evaluation:

Vanderbilt correctly kept their No answer and were able to better justify the answer following collaboration.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.11.4 Question 037 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Access to a voter's profile is done through a GET request to /profile which can only be done after a successful login from the /login webpage, which requires a valid username/password combination. The logic for this lies in webcontroller.manager.LoginManager.handleParcel, which in turn uses webcontroller.manager.TokenedPasswordChecker.processPassword to verify the login credentials.

The TokenedPasswordChecker class restricts valid passwords to be between 5 and 15 characters in length (inclusive) and contain only characters in the ASCII range [! , ~], as well as not containing the space character. The processPassword() method checks the validity of the password while building a token which combines some elements of the secret password with the given password. This token is then compressed and is sent back to the user in the event of an incorrect password. […] If the token contains all the characters from both the secret and given passwords, then it would be possible to mount a prefix attack (e.g., CRIME) to recover the true password. However, only some of the characters of the secret password are included, due to the behavior of the pad variable. When pad = true the current character in the secret password will be added to the token string. For example, if the secret password = "abcdefghijklmno" and the given password = "123456789abcdef" then it is possible for both of the following cases:

token = " abcdefghij 123456789abcdef"

token = " abcdfghijlo 123456789abcdef"

Essentially, the value of pad is true for the first several characters, depending on the length of the true password. Afterwards, pad is true only 50% of the time. For the following lengths, pad will be true for some number of iterations: […]

Consider the voter with username = jose4district4.gov and password = district4Judge. Using the packet analysis portion of our tool Profit, we analyzed the size of the packet containing the token string for a sequence of incorrect password, with each successive attempt matching one more character than the last: […]

As more characters are matched, the packet size seems to decrease, but not always. Due to the randomness of the password checking method from above, an averaging of several packets at each step is required in order to decrease the amount of "noise" in the size of the packets. In the worst case of a password of length 15, an attacker would need to guess ~1400 operations if averaging of packets was not required. In this case, since multiple packets must be obtained at each step, the operational budget will be greatly exceeded. Thus, while there is a space side channel, it does not fit within the budget."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.11.5 Question 038 (AC Time, Unintended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"It does seem that the server tries to connect to other few servers which contain fragments of data, and then processes those fragments of data such as gathering those data into one view. The server attempts to connect to other servers for three times if any connection failure happens. This reconnection takes some time for the server to do, but we found that within the input budget, we cannot make server to run for over 300 seconds. We thus resorted to other possible source of vulnerability of the server. The server uses Math.random at several points, one of which is like below […]. Note: that the Math.random() is not cryptographically secure and it takes current time as its random seed. It seems that if an attacker can estimate the time when to do the attack, which will cause the Math.random to generate a value that is greater than 0.6 or 0.5 for several times, then the above loop will run for long because the value q will not be updated (by the statement ++q in the inner loop). We tried to randomly run our attack at various different points of time, but we did not succeed in exposing the vulnerability in question by doing so."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

“Northeastern, Colorado, and Grammatech all found a vulnerability they could exploit involving the validation of registration keys, though initially each of their attacks were well over budget (~6KB). They’re intuition was that this version of simplevote validated registration keys in a similar way to the simplevote from E5, and that version was vulnerable to an algorithmic complexity timing attack. Registration keys in simplevote_3 are 12 characters long, with 6 characters corresponding to an integer value,N, which must be divisible by 7. The validation of registration keys involves, among other things, computing the recurrence relation F(n) = 4*f(n-1)*(1-f(n-1)), though in a very convoluted way. This means that registration keys corresponding to higher values of N take longer to validate. The key observation the 3 teams made was that intermediate values of N were stored in a cache in intervals of 7, with the first ~142 values > 0 (corresponding to values of N < 995) precomputed and already stored in the cache. Additionally, registration keys with higher values of N take longer to validate. The 3 teams attack involved sending 6 packets with handcrafted registration keys corresponding to values of N = 995, 996, 997, 998, 999, 1000. Since 1000 is the highest value of N in this range, it should take the longest to validate. Indeed, when this registration key is sent to the server, the server hangs for > 300s. As mentioned above, however, this attack is well over budget. Sending each registration key to the server takes around ~1KB, with the whole attack being 6-7KB in size. However, during the live engagement, Northeastern was able to get under the budget by creating a single SSL tunnel and sending manually crafted HTTP requests containing the registration keys to the server, then closing the SSL tunnel. Crucially, the TCP session was never dropped or lost; in other words, the entire attack occurred during a single TCP session. The total size of the packets sent was ~3KB in size. Some of us attempted the attack and exhibited the server blocking the validation of other registration keys for the following times:

Colorado : 7min 17s

Utah: 7min 20s

Vanderbilt: 7min 23s

Grammatech: 7min 18s

Iowa: 7min 17s

We have submitted the attack script (from Northeastern) with our submission. The attack is under 038_exploits/attack.py and should be run by the attacker. The pcap file is attached as well. While we were not able to do so during the live engagement (due totime constraints), our tool JAnalyzer would have been helpful in narrowing down which snippets of code contained the most suspicious elements (e.g., finding all the loops and narrowing down based on the loop levels).”

Live-Collaborative Evaluation:

Following collaboration, Vanderbilt reported the same unintended vulnerability as their fellow collaborators, one that derives from the take-home responses of Northeastern and Utah.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.12 SimpleVote 4

A.6.6.2.12.1 Question 023 (SC Time/Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Each ballot is associated with a registration key. In order to view/change/submit a ballot, the corresponding registration key must be provided. This is done in /registrationkey webpage upon clicking on an election. The code which handles this logic resides in the class straightselection.guide.SanctionTestGuide.java, mainly in the handleDelivery() class.

For a registration key to be valid, it must conform to a complex specification. This is done through the confirm() method in the straightselection.SanctionAuthenticator.java class. There are several validation checks: The registration key must be of length 12. The first 6 characters must be digits. The last 6 characters can be digits (other than 3,6,7), but may contain the characters 'A', 'C', or 'E', which in turn get translated into 7,6,3, respectively.

The key is then split into two parts: kStr is the first 6 characters, nStr is the last 6 characters. Cast into an Int, nStr must be divisible by 7 and in the range [0, SanctionAuthenticator.MAX_N]; SanctionAuthenticator.MAX_N is initally 995, but increases by one each time nStr is larger than it. The last phase of validation is the most complex, occuring in the method SanctionAuthenticator.confirmValue() […] Assuming an attacker could retrieve the registration key from the victim voter, another check must be passed. Each voter is given a unique ID associated with their profile, and each registration key is tied to this ID. Once the registration key is checked for validity in SanctionAuthenticator.confirm(), the application cross references the current user's ID and registration key with known ID/key pairs for that particular election. This is done in straightselection.StraightSelectionServiceActual.confirmSanctionKey(). Thus, it is not sufficient to simply get the registration key. Either an attacker must somehow change their identity, or add their ID/registration key pair into the application's database. Given the actions an attacker can make as a voter, this does not seem to be possible."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.12.2 Question 030 (AC Time, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"The server for Simplevote_4 also tries to connect to other servers to gather all the data fragments into a single view. A connection attack was also attempted on this program to see if it exhibited the same issue as Powerbroker on an earlier challenge, since the server makes multiple attempts to connect to the other servers in the event of a connection failure. This did not impose

enough delay to cause the benign request to be delayed greater than 300 s. Similar to Q038 for Simplevote_3, we believe this does not have a temporal vulnerability."

Take-Home Evaluation:

The challenge contains an intended vulnerability in triggering the application to clear the cache used to process registration keys.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"GrammaTech and Colorado found a vulnerability they could exploit involving the validation of registration keys. They thought that this version of simplevote could be vulnerable to an attack similar to one used on simplevote in E5: sending multiple handcrafted registration keys to the simplevote server caused the server to hang for long periods of time due to the underlying validation algorithms. Registration keys in simplevote_4 are 12 characters long, with 6 characters corresponding to an integer value, N. The validation of registration keys involves, among other things, computing the recurrence relation F(n) = 4*f(n-1)*(1-f(n-1)), though in a very convoluted way. Both teams noted that intermediate values of N are stored in a cache (Stash.java), and that once the cache becomes full, ten items from the cache are deleted; this is done through Stash.considerAdding(), which considers adding recently computed values of N to the cache, and Stash.composeOpening(), which clears 10 items from the cache. Both teams convinced us that this is vulnerable to a complexity attack, and that by submitting a registration key corresponding to N=995 three times, the registration verifier (SanctionAuthenticator and ChaoticOperator) would then stall for some time (> 300s). Specifically, the first time sending the key increases the size of the cache, the second time sending the key clears ten items from the cache, and the third time sending the key stalls the authenticator for some time. Each sending of the registration key is ~1KB, with the whole attack being ~3KB, well under the budget. While we were not able to do so during the live engagement (due to time constraints), our tool JAnalyzer would have been helpful in narrowing down which snippets of code contained the most suspicious elements (e.g., finding all the loops and narrowing down based on the loop levels)."

Live-Collaborative Evaluation:

Following the collaboration with GrammaTech and Colorado, Vanderbilt identified the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.6.6.2.12.3 Question 039 (SC Space, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"Since an attacker does not have admin privileges and it is unlikely for the attack to go on for multiple hours, the best avenue for an attacker to glean any information would be to modify a ballot in a way which leaks voting information, and update said ballot ten times in order to trigger an update. The logic for updating ballots resides in EditBallotGuide.handleDelivery(). A ballot has a number of questions, some of which are multiple choice with the rest being free response. Of particular interest is a snippet of code which parses the free response and legitimizes the choice: […] That is, if the free response starts with whitespace, a handmade trim() method is invoked: […] When trim is invoked, ResultsMessageBuilder.update1() is called

before the trim, and ResultsMessageBuilder.updatel() is called after the trim: […] These two methods work together to define a the INCLUSION_THRESHOLD which determines which ballots should be included in the election results message. Once a ballot is updated ten times, AggregationService.gatherResults() is invoked, which in turn calls ResultsMessageBuilder.buildMessage() to build the results message. During this process, responses are parsed and legitimized further, only being included if they are completely necessary […] Once the message is built, the message is compressed by first building a Trie and using an unusual form of LZW compression on said Trie. We used the package analysis portion of our tool Profit to analyze what would happen if an attacker attempted to craft responses which manipulated what was included in the election results message. Specifically, we in one of the free response answers, we put the specified candidate with two spaces before the name, and a number of spaces after the name, in order to see if there was any change in the compression scheme which would affect the resulting size of the election message. The number of spaces increased by 5 for each interaction, and we tried both valid and invalid candidates […] The interaction corresponding to the number of spaces is on the X-Axis, with the total packet size for the interaction on the Y-Axis […] While there is some size information, it is identical regardless of whether the candidate was a valid or invalid candidate. This does not seem to be a side channel"

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.12.4 Question 041 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"To view a voter's profile, a GET request to /profile must be done, but this requires a valid sessionId cookie. This is given upon entering valid username/password credentials to the /login page. Given that the question assumes we know the username of the voter, it seems likely that the password is the secret. The bulk of the /login logic resides in the LoginGuide.handleDelivery() method. Once a username/password combo is submitted, the given password is compared to the correct password (assuming the username is valid. This comparison is done using a method call to Member.matches(), which subsequently compares the given username and the correct username, and the given password with the correct password using the method passwordsEqual(): […] This method seems to leak timing information regarding the length of the secret, but not necessarily the individual characters. Additionally, there seems to be no restriction on the length of the password, nor the contents of the password. This does not appear to be a timing side channel. Aside from getting a voter's password, an attacker could gain access to the voter's profile page by hijacking their session id. Once a voter successfully logs in, the server assigns the voter a cookie with a random UUID as a session id. Since the voter is

guaranteed to log in once, an attacker could view the encrypted packets which contain the cookie (once the voter logged in) but it is not clear how an attacker could see the plaintext cookie."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.13 STAC Coin

A.6.6.2.13.1 Question 007 (AC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"We began by gaining familiarity with the program and noticed there are separate threads for the miner and REST end points. There is only one miner thread and a new thread is spawned for serving each end point whenever a request is made. We identified, Miner.mineBlock(), Miner.tryToUpdateFromPeer(), BlockChain.extend(), BlockChain.extendFromTip() as suspected methods that may allow the response time for /balance/me to exceed 100 secs. We modified the decompiled code and added a block containing an infinite loop in the methods mentioned above in order to test whether or not we could block the benign request. However, even with such a manually crafted infinite loop and up to 100 requests, the benign request was not blocked. We observed that whenever a new balance/me request is received, a new thread is created to serve that request, and thus believe that there is no vulnerability present."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"Iowa State described an exploit that triggers the vulnerability. They said that they looked for thread related vulnerabilities because it "felt suspicious." In our original analysis (above), we described how we also suspected these, but ruled them out when our manual modifications could not block the benign request even with 100 requests that were manually modified to spin in an infinite loop.It turned out that the maximum number of parallel threads that stac coin's web framework allows is 200, and so if you send 400 spend requests from a malicious peer to the benign clients, they will in turn send more than 200 requests back out before proceeding. If you modify the client to not respond to those requests, then the benign request cannot be serviced since there are already more than 200 outstanding threads.However, this is based on the assumption that the benign user sends all these requests. This is not a valid assumption, and so we believe that this is not a vulnerability."

Live-Collaborative Evaluation:

Following collaboration, Vanderbilt kept their correct answer of No. They were not convinced that the other teams made a valid assumption for their vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.13.2 Question 008 (AC Space, Intended Not Vulnerable, Take-Home: No, Live: No)*

Take-Home Response:

"When either of the two peers detect an out of sync block chain, the Miner.trytoUpdateFromPeer() method is invoked, which eventually calls Persistence.writeWalletToDisk() method if valid blocks are identified and added to the miner's blockchain. The out of sync detection occurs during initialization and after every time a miner announces a recently mined block. A miner starts mining if it receives a valid transaction from the network. There are two ways in which a new transaction request can be made: (1) using following the spend endpoint, and (2) using the newTx endpoint. The spend endpoint consumes less input budget but is password protected. The newTx endpoint is not password protected but consumes more budget as it includes complete transaction details. After receiving a valid transaction, a miner starts mining, and whichever miner mines the block first announces the block and adds the block to its chain. One exploit that we attempted was to create a transaction and send it directly to the target. However, this exploit only increases the wallet size by an amount that is almost twice the bytes sent by the attacker. For instance, bytes transferred during one transaction were 483 bytes and the size of the wallet.dat file increased by 976 bytes. Another exploit that we attempted was to send a block to the target using the newBlock endpoint. However, we found that this exploit was also unable to exceed the budget. Moreover, Persistence.writeWalletToDisk() is preceded by the Blockchain.extend() method, which only adds specified blocks to the block chain; manual inspection led us to believe they are bounded in a way that cannot be used to exceed the budget."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

"GrammaTech stated that they believed that the way the question is phrased implies that the input budget isn't guarded by the total amount of traffic sent by the malicious user, but rather what initiated that traffic: if the traffic is generated by the malicious user in response to a benign user's request, then it doesn't count against the input budget. They convinced us that this is a reasonable interpretation of the original question, in which case it is possible for a malicious user to compute a really long chain entirely offline (which may take weeks of computing time for a > 1GB chain). After this really long chain is computed and the malicious user is connected to the network, the benign client will try to update from its peer and request this (really long) chain, which will then be sent by the malicious user to the benign client as a response. This will then be written to the wallet.dat file by the benign client and will exceed the resource usage limit since, by design, the chain computed offline exceeded the resource usage limit."

Live-Collaborative Evaluation:

As correctly noted by Utah in their response, the input budget applies to the total bytes sent from the attacker to the device with the application under attack. The reported vulnerability is believed

to exceed the budget. We note that GrammaTech's interpretation that bytes sent in response did not count towards the budget was the motivating factor for the "Yes" response; therefore, this response is treated as a "No" response. The challenge does not contain an intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.14 Swappigans

A.6.6.2.14.1 Question 002 (AC Time, Intended Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

"We began by trying to understand the program and experimenting using the web-based interface. We then loaded the program in the Janalyzer, which reported three recursion groups, two of which are in the SwappigansItemMatcher class. We thus began investigating this class in more detail. The flow of control within SwappigansItemMatcher is complicated to manually debug, so the approach is to take a few steps back and examine control flow branches that affect the size of the dataset being operated on, and can be influenced by the attacker. Surprisingly, there appear to be only two […] T is the value of the item the user desires to purchase and this code simply removes groups of items whose collective value exceeds the target value. The items are grouped together using the algorithm and as a user we only have very indirect control over the grouping, but the rule of thumb is that if we don't want items to be removed from the data structure, we should ensure that it's difficult or impossible to create an item group that exceeds the value of the desired item. This, as it turns out, doesn't actually matter. […] The factor that stands out is that based on a condition, an item is added to a working dataset which is then further operated on by the algorithm. More data more problems, as the old proverb goes, so let's try to make sure the data structure contains as much data as possible. In order to bias this condition toward true, we obviously have to lower the delta parameter. Is this parameter under the user's control? Turns out it absolutely is. Through the many layers of recursion and indirection, delta is passed unchanged from the algorithm invocation point: the method calc. It is defined as: `precision / (2 * this.items.size())`. Where precision is a parameter passed to the invocation of calc. To make delta smaller, we can increase the size of the user's item knapsack. Let's look at the ItemPurchase class to see how the precision parameter is defined. […] To make precision smaller, we can increase the price of the item to be purchased. The values itemPrice and SwappigansItem.ITEM_MAX_PRICE are integers, and the result of this division will always equal 0 (making the precisionalways equal 0.15) unless itemPrice is equal to SwappigansItem.ITEM_MAX_PRICE. So we must increase the price of the item to be purchased to the maximum possible price ($1,000.00). […] The statement yi.price > last.price in the algorithm, based on the variable name last, provides us with the hunch that the difference in price between items should be small in order for the if condition to be true. In this case, all items differ by one cent. Practical testing shows that this hunch is correct. NB: for an unknown reason, the actual runtime also varies with the price of the first item (i.e. the sequence of prices "101, 102, 103,..." exhibits a different runtime from the sequence "201, 202, 203,...")."

Take-Home Evaluation:

Vanderbilt did not discover the intended vulnerability that required the attacker to exceed the price bound to exceed the resource usage limit. Instead the team similar to several other teams has identified a potential unintended vulnerability where the input budget is used to add a large number of items with minimal price difference between them before attempting to purchase the most expensive allowed item of $1000.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly identified an unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.14.2 Question 020 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes, Live: Yes)

Take-Home Response:

“We want to impersonate a user. We know the username, but not the password. There are two ways to achieve this:

1. Extract the password of the target user, OR
2. Extract the session key of the target user

The login request allows a user to provide username and password and obtain his/her session key. It does not really "log you in" in any stateful way -- it just trades you your session key in exchange for your password.

For all other requests, no password is needed, and the session key suffices for the server to honor the request. […] A user's session key is merely his/her username, AES-encrypted with the server's master key. The server's master key is created when the server is booted, and lasts for as long as the server is alive.

Thus, a session key is not really an identifier for a client session in the usual sense. Rather, it is an identifier for a certain username within the current session of the server. (If the same user connects from home and from work, using different browsers, or at different times of day, or on different days, he or she will still have the same session key, as long as the server has not been rebooted.)

Passwords, on the other hand, are concatenated with a salt string and then their concatenation is MD5-hashed to obtain a HSP (hash of salted password). The username, HSP and salt of each user are stored on the server. Passwords are not. Users can be created either at boot time from the users file (in this case username, HSP and salt are read in from a file) or dynamically during runtime (the register request). In this case the client provides a (fresh) username and password and the server will generate an 8-char salt, concatenate, MD5-hash, store the username, HSP and salt, and throw away the password.

Both the server's master key and each of the dynamically-created user's salts are generated by the same algorithm, getSaltString(int n), where n is the desired number of characters.

- This algorithm uses a simple Random instance created from scratch on each call to the method, and seeded with currentTimeMillis() / 1000L. Thus, any invocations of getSaltString(n) within the same one-second period will return the exact same string.
- Also, the character set is small, so the actual entropy of the generated keys is well below the optimum one.

- For example, a master key has 16 characters, and is therefore used as a 128-bit AES key. However, since they are all [A-Z0-9] characters, we know that the first bit of each byte is guaranteed to be a zero.
- Moreover, 128 bits would normally mean 2^128 combinations, but the fact that they stem from 16 characters chosen out of a 39-character alphabet means that the brute-force space is reduced to 39^16 possible combinations. (Still too much to brute-force, but significantly less.)
- Using active operations, you can create new users in a small fraction of a second. Thus, if you create many users within the same second, they will all have the same salt!
- On encryption and decryption of usernames/session keys, Cipher objects are always used after being re-created from scratch, and are initialized without any fresh IVs or other source of randomness.
- Also, the RestrictedSSLServerSocketFactory restricts the cipher suite of the SSL channel to TLS_RSA_WITH_AES_128_GCM_SHA256, which is considered weak. (But we do not believe that cracking the SSL layer is the goal here.)

Side channel in space: length of password

We used Profit to record and analyze login traces using almost-identical usernames (all of same length) but with very different passwords (of varying lengths).

Profit immediately revealed a side channel in space (the size of a certain packet) that lets the attacker extract the length of the password provided by the benign user during login. If the username is known, this side channel lets us infer the exact size of the password as long as the password only contains basic alphanumeric characters. (If the password can also contain spaces, symbols, etc, then some characters may need to be urlencoded, and then instead of an exact value we would get an approximation, as some characters would count triple.)

[…] We ran analyses for several sets of passwords on successful and unsuccessful login requests. Profit did not find any clear side channel in time that would let us infer the password. This may be caused by our input sets not being the appropriate ones.

What can be timed during the login process: If we want to extract the password, the only request that actually handles passwords is the login request. What can we expect to obtain when timing executions of the login request? The two main sources of time expenditure inside the request handler are the following two lines: […] So, in the best case, we can expect to obtain a time proportional to the sum of [the time it takes to compute MD5(password+salt) and compare it (using String.equalsIgnoreCase()) to the stored hash and the time it takes to AES-encrypt the username using the master key.] For the purposes of extracting the password, our Profit analysis did not reveal any interesting correlations between the password and the time taken by the sum above. The potential prefix (segment oracle) side channel in comparison of the strings does not look promising because (i) it is not amplified, and hence would probably be too small to be detectable on the wire, and (ii) the comparison is performed on the MD5 hash of the salted password, not on the password itself.

Extracting the master key: If you can infer the master key, you can infer any user's session key (you just need to AES-encrypt its username with the master key). So how do we find out the server's master key? Sadly, we don't know the time at which the server was booted: if we did, it would be trivial to use that as a seed for Random and reconstruct the master key. However, thanks to the whole-second granularity of the seed, the search space of potential moments in time

(i.e., whole seconds) in which the master key might have been created is astronomically smaller than a brute force on the key itself. There are only about 35 million whole seconds per calendar year. Thus, we propose to try and guess when the server was booted, starting from the current time and going backwards into the past, second by second, until we find a key that behaves just like the current master key […]"

Take-Home Evaluation:

Vanderbilt flagged the same unintended out-of-scope vulnerability as several other teams. A key assumption of this vulnerability is the attacker knowing when the server was started. In future challenges we will work to clarify that the attacker does not know when the server was started.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly identified an out-of-scope vulnerability.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.14.3 Question 031 (SC Time, Intended Not Vulnerable, Take-Home: No, Live: No)

Take-Home Response:

"The attacker wants to discover which item is purchased by a target user, but does not know what items are in the target user's item list. The number of items in the target user's inventory (and their prices) can affect the timing profile of his purchases. In the worst case, the target user could be any possible user. Thus, showing that we cannot do this within the allotted budget for someuser would suffice for a negative answer. The name and ID of the item to be purchased are not relevant for the purposes of choosing a bartering subset. Thesefore, once the target user is set, the only free parameter remaining is the price of the item to be purchased.

In order to find the most-leaking side-channel observables in time, we conducted the following Profit experiment. We selected 10 items with widely different prices, well spread across the price range. To do this we split the set of items that RAGUSA can purchase into 10 price ranges ($0-$100, $100-$200, ... $900-$1000) and picked the most expensive item in each range.

Using Profit, we recorded user RAGUSA purchasing each of these 10 items, 200 times per item, for a total of 2000 recorded traces. Profit's automated analysis returned the following ranking: […] Our next goal was to confirm the existence of a set of more than 32 items that cannot be reliably separated by these observables, even with considerable offline profiling, so that the available oracle queries would not be enough. Since the previous experiment was affected by a phenomenon that looks like warmup (see figures above), this time we ran a stronger warmup phase (about 1000 traces' worth of purchases without recording them) before running and recording the real ones. This mitigates the artificial downward curve that can be seen at the beginning of previous experiments.

We selected user HULTBERG, because it is the user whose least-expensive item is most expensive wrt other users. We used Profit to automate the purchase of the 100 least-expensive items available to that user (these 100 items range in price from $1 to $100.61). After warmup,

we sniffed HULTBERG purchasing these 100 items 50 times per item, for a total of 5000 recorded traces.

Profit reported essentially no leakage (no observables above 3% leakage): […] The last observable yields a small amount of leakage (visibly higher mean for the first few products than for the last few). However, let us recall that this experiment is 100 items wide, whereas we only have 31 oracle queries left after sniffing the benign interaction. In the central part of the last plot, for trace numbers going from approx. 2000 to 4000 (that is, from the 40th to the 80th item), we can see complete overlap of the observables that is at least 40 items in width. This is clearly not enough to yield 85% probability of success within any subset of 31 items."

Take-Home Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable. While this has no impact on the evaluation of this response, one potential avenue for attack that appears un-explored is the ability to apply the restrictions on the subset of items that can be used to purchase each item in the canned data. This can be used to reduce the space of the secret and create a potential mapping. Additionally, if there existed a path by which an attacker could determine what user owns what item, this could further reduce the space. The presented experimental results instead attempt a purchase of a subset of items demonstrating that if the subset of items cannot be distinguished within the operational budget then there can exist no side channel which is effective in demonstrating the lack of a passively persistent side channel.

Post-Take-Home Analysis Ruling: Correct

Live-Collaborative Response:

Same as Take-Home.

Live-Collaborative Evaluation:

Vanderbilt correctly concluded that the challenge program is not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.6.6.2.15 Tollbooth

A.6.6.2.15.1 Question 011 (AC Time, Intended Vulnerable, Take-Home: No, Live: Yes)

Take-Home Response:

"So 5 kB input budget = 5120 bytes = appx 9 of the shortest commands (GET index.html) and 4 of the longest (POST add_value). So the command input limit is between 4 and 9 commands.

Looking for time consuming processes I found 4 places that call Thread.sleep, but none are of interest in creating a vulnerability […] Other time-consuming processes can involve BigInteger or BigDecimal math (there are none) or possibly just floating point values that loop on an equality instead of greater than or less than qualifiers. There are no uses of float or Float and none of double. The only uses of a Double value is in handling WebInterface.do_add_value which passes the value to TollBoothRootWorker.addTransponderValue if the value was > 0.0 or to TollBoothRootWorker.transferBalance otherwise. addTransponderValue will add the numeric value to the amount entry in VehicleRecord for the current transponder, whereas transferBalance will subtract the value from the current transponder and add it to the transfer_id VehicleRecord specified. The only comparison performed on this VehicleRecord.amount entry is simply testing if > 0.0. No complex math seems to be used anywhere, only simple addition/subtraction of

Double values. There are no limit checks performed on the value added to or subtracted from the VehicleRecord.amount entry, so if this was used in a loop, this could be a vulnerability, especially since the user is specifying this value. However, this does not seem to be the case."

Take-Home Evaluation:

The challenge contains an intended vulnerability that enables an attacker to create and submit a malformed packet that causes the root tollbooth node to fail to respond to a charge request.

Post-Take-Home Analysis Ruling: Incorrect

Live-Collaborative Response:

"GrammaTech identified a suspicious area in PacketQueue.sendPacket() that appeared as if it would broadcast the incoming TollBoothPacket if the destination was null. It was suggested that we try to manually modify a client to implement this attack. When this was tried (a manually modified malicious leaf node sent a packet with a null destination), we observed that the root tollbooth worker thread crashed (because of a null pointer exception in TollBoothWorker.run()). Because this thread crashed, the benign request in the question will not receive a response within the 500s resource usage limit. Thus, we felt that, although this is a "crashing" exploit, that it fell within the scope of an algorithmic complexity vulnerability in-line with the way the question was phrased."

Live-Collaborative Evaluation:

Vanderbilt reported the same unintended vulnerability as several other collaborators.

Post-Engagement Analysis Ruling: Correct

A.6.7 Two Six Labs (Control Team)

A.6.7.1 Two Six Labs Overview

Two Six Labs out performed all the R&D Teams during the take-home engagement with an 89% accuracy and 38 of the 43 questions answered. As a result of this performance the team was excluded from the live collaborative engagement to allow the R&D Teams to attempt to match the control team's performance by working together. Two Six Labs tied for the highest TPR (75%) and had the highest TNR (100%). Against the segmentation of all questions (unanswered questions treated as incorrect) Two Six Labs outperformed the R&D Teams with the highest accuracy of 79% and the highest TPR of 71%. The team's TNR in the all questions segmentation decreased from 100% to the third highest of 85%.

A.6.7.1.1 Take-Home Engagement Scores

Table A-141: Engagement 6 Two Six Labs Take Home Engagement Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	8	7	88
SC Time	12	12	100
SC Space/ Time	3	3	100
AC in Space	7	5	71
AC in Time	8	7	88
Total	**38**	**34**	**89**

Table A-142: Engagement 6 Two Six Labs Take Home Engagement Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	16	12	**75**
Not Vulnerable	22	22	**100**
Yes Answer	12	12	**100**
No Answer	26	22	**85**

A.6.7.2 Two Six Labs Specific Responses

A.6.7.2.1 BattleBoats 3

A.6.7.2.1.1 Question 022 (AC Space, Intended Vulnerable, Take-Home: Yes)

Take-Home Response:

“There is a loop in the HitPinpointer.determineHitTime() method which, after 25 iterations, begins adding entries to the trajectory HashMap on each successive iteration. This code is called when an opponent makes a shot. We determined experimentally that the number of iterations through this loop significantly increases by sending a particular combination of a very small height, initial velocity, and initial angle. An attacker can drive the memory usage of an opponent above 1 GB by issuing: 'shoot .0000000000000000000000000000000000000001 .0000000000000000000000000000000000000001 .0000000000000000000000000000000000000001 0'

The size of the shoot message sent to the opponent is well within the input budget. Our analysis indicates that over 1 million instances of BigInteger are instantiated during this loop, largely due to the number of calls to HitPinpointer.height() which allocates 5 new BigDecimal instances (and by extension, 5 new BigInteger's since each BigDecimal contains a BigInteger) each time that it is called. We note that there may be another vulnerable portion of the code that contributes to this AC attack: we observed OceaScreen.pullOceanSquares() instantiates a large number of BigInteger objects as well, but it pales in comparison to determineHitTime(). Since we had a working exploit, we did not spend any more time confirming which other portions of the program might be using a lot of memory as a result of our attack.”

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.1.2 Question 026 (SC Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“We performed an exhaustive manual search of the code base. We examined all locations where the opponents boats are used in a computation. We confirmed that no messages sent to an opponent contain the boat locations directly. We considered the possibility that a message might vary based on the location of ships. Returning information in response to an opponent's shot was the only location we identified of any interest with respect to this question.

[…] our analysis indicates that information gained is limited to the 1-4 squares of the board where a shot landed. We did not identify any way for a user to learn their cannon position in a space side channel, which would enable them to brute force shots that always land on 4 squares on an opponents board. Our conclusion is that there is no side channel in space from which the attacker can discover the coordinates of all the opponent's boats on the ocean board.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.1.3 Question 040 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“[…] We considered the possibility that the timing of a sent message might vary based on the location of ships. Returning information in response to an opponent's shot was the only location we identified where there could be a side channel in time because the opponent automatically sends a response after calculating whether our shot was a hit or a miss, and this computation could theoretically take longer or shorter depending on the location of the opponent's ships. However, the ships are only ever used in a few places during this computation. Notably, all boats are checked in a for loop to determine whether a shot landing in a square has hit a boat. This occurs in OceanScreen.tagHitSquare(). If instead, only the boats located near a shot were examined to see if they had been hit, then there would be greater chance for a timing side channel to leak the locations. We did not find any operations on the ship locations that would take a variable amount of time. Our conclusion is that there is no side channel in time from which the attacker can discover the coordinates of all the opponent's boats on the ocean board.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.1.4 Question 043 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“[…] We considered the possibility that the timing of a sent message might vary based on the location of the cannon. Returning information in response to an opponent's shot was the only

location we identified where there could be a side channel in time because the opponent automatically sends a response after calculating whether our shot was a hit or a miss, and this computation could theoretically take longer or shorter depending on the location of the opponent's cannon. In particular, we looked closely at whether computation time might differ depending on whether a shot landed on or off the ocean board. Such a difference in timing would allow a player to determine how far away from the edge of the board their cannon has been placed, revealing the cannon's coordinates on the ocean board. The OceanScreen.pullOceanSquares method calculates whether a shot lands on the ocean board, but the computation is only cut short if tooFar also evaluates to True. This removes the possibility of any information leakage, because a shot is only tooFar if it has no chance of hitting the board no matter where the cannon is located, which the player taking a shot would already know. Similarly, there will be more time taken to respond if a shot hits multiple squares, however this information can also be determined by the player taking the shot in advance, and does not reveal any information about the cannon location. To test our static analysis, we timed how long it took to receive a response to various shots, and did not notice any changes in the timing of shots based on the cannon's location, or whether or not the shot landed on the ocean board. […]"

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.2 BraidIt 3

A.6.7.2.2.1 Question 004 (AC Space, Unintended Vulnerable, Take-Home: Yes)

Take-Home Response:

"The only file that is generated as a result of executing the program is the log file. This is appended to during normal operations of the program, however the largest volume of messages are generated in the braid reduction phase of operation (once a user has made a guess and are reducing it with the modified braid).

In plait/Plait.class there is a potentially infinite while loop[…] If there is a condition that forces isReduced() to always return false then the loop is infinite. […] We use the following braid as the malicious modified braid in our attack:

"NOPQRSTUVWXYZZZZZZZZZZZZZZZZZZZZZZZZZZnmopqrstuvwxC"

As long as the braid being compared against doesn't contain the letters "a","b" or "c", the loop will be infinite and cause the log file to continue writing more lines. […]"

Take-Home Evaluation:

The potential unintended vulnerability identified by Two Six Labs was confirmed.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.2.2 Question 027 (SC Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"[…] There is no side channel in space where a player can determine which braid was modified. […] there is no input we can provide that influences the initial braids. When we receive the

modified braids, there is padding added to the modified braid, but the padding is equal to the modified braid length, which does not reveal any new information. […]

There is still a totally offline attack where one can run the braid equivalence against the modified braid and all candidate braids, but this is not a space side-channel and fails when two of the initial braids chosen are equivalent, which again we cannot influence."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.2.3 Question 029 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"[…] there is no possible timing side channel that would reveal the contents or lengths of these braids outside of breaking the encryption. […] In both the RSA and the Diffie-Hellman logic there are indications of weaknesses that would allow for a timing side-channel, however each of these are mitigated with known defenses. In the RSA decrypt method in CipherPrivateCode the logic would be vulnerable to the "Montgomery reduction" timing attack, except the ciphertext is pre-multiplied by a random blinding factor r, which prevents the attacker from determining either of the prime factors by sending crafted ciphertext. The "generateMasterSecret" method in CodeExchangeServer (which implements Diffie-Hellman) makes calls to the ModularPower modularPower method. This method implements a Montgomery Ladder for performing modular exponentiation. The implementation assures that the potentially expensive fastMultiply is executed in the case that the secret key bit is either 0 or 1, which thwarts a timing attack. Since the "secretCode" passed into "generateMasterSecret" is generated from 8 bytes of SecureRandom, the generated AES key is secure from brute force."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.3 BraidIt 4

A.6.7.2.3.1 Question 032 (SC Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"Similar to braidit 3, the attacker has no influence over the braid lengths, and the braids are randomly generated by the opponent locally. When the braids are sent, the modified braid is padded according to its length multiplied by its original position amongst the braids. This would lead to a space side-channel except the braids are all randomly shuffled before being sent back to the attacker, so the original index of the modified braid reveals no information about the current index."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.4 Calculator 1

A.6.7.2.4.1 Question 014 (AC Space, Unintended Vulnerable, Take-Home: No)

Take-Home Response:

"[…] As calculations on `GreatNumber`s are generally memory efficient (ie not more than approximately linear with coefficients small enough that a blow-up from 60 kB to 4 GB is not plausible) it is not possible to cause the server to consume 4 GB of memory via the intermediate products of calculations. More relevantly, the attack detailed in the response to Question 009 does not work against the challenge program. It is impossible to send a literal number of more than approximately 10,000 digits to the challenge program due to the 10 kB limit in `MultipartAssistant::HIGHEST_PARCEL_SIZE` and the `if (obtainDegree() > 50000 / (10 * exponent))` check in `GreatNumber::exp()` prevents the construction of numbers that push the 50,000 digit limit in the `GreatNumber::GreatNumber(int[], int, boolean)` constructor. While it is possible to increase RAM usage by sending multiple requests to multiply ~10,000 digit numbers by single digit numbers, performing a modified multipart version of the attack from Question 009, this attack requires ~10 kB of PTU per request, allowing only 6 requests within the Input Budget, and forces the challenge program to consume only ~300 MB of RAM per requests, resulting in a total maximum effect of ~1.8-2 GB of RAM, which is well below the target of 4 GB."

Take-Home Evaluation:

The challenge program does not contain an intended vulnerability; however, Colorado and Northeastern identified an unintended vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

A.6.7.2.4.2 Question 016 (AC Time, Unintended Vulnerable, Take-Home: Yes)

Take-Home Response:

"[…] The vulnerability lies in the fact that multiplying large numbers takes a relatively long amount of time. By chaining multiple such multiplications (in this case, `9^5000*9^5000`) into a single request, the runtime of the request may be inflated arbitrarily and increased linearly in the number of multiplications. The multiplications are chained together with alternating additions and subtractions to prevent any of the intermediate values from growing above the limit of 10^50000 implemented in the `GreatNumber::GreatNumber(int[], int, boolean)` constructor. It is believed that multiplications are slow due to the complex operations performed in `GreatNumber::transformMultiply()`. The multiplications are split into two batches across two different requests to avoid exceeding the 10240 byte request limit in `MultipartAssistant::HIGHEST_PARCEL_SIZE`. This does not, however, inhibit the attack, as the server is single threaded and only processes one request at a time in the order that requests are received, so both exploit requests are processed before the benign request."

Take-Home Evaluation:

Two Six Labs identified a potential unintended vulnerability that was confirmed.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.5 Calculator 2

A.6.7.2.5.1 Question 003 (AC Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"[…] The only file that the challenge program produces is a log file. This file is only accessed in the AbstractHttpCoach class, in a call to Future.get(), a java function call. The only times the log file can be written to are when the get() receives either a thread interrupt or an execution exception. None of the custom exceptions implement either of these exceptions, and neither exception is thrown by any of the in-scope code. Even if there were sequences of operations that could trigger these exceptions, there is no indication that these warning messages would give the required file blowup to solve the question under the input budget."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.5.2 Question 005 (AC Time, Intended Vulnerable, Take-Home: Yes)

Take-Home Response:

"Yes, there is an algorithmic complexity vulnerability in time that causes the real runtime of the benign request to exceed the resource usage limit given the input budget. When the server is sent a specifically crafted packet the benign request will not receive a response until after the response to the exploit request, which will take over 120 seconds. As the exploit request requires <2 kB of payload the total PDU used is, even with TLS overhead of ~1 kB per request, well below the 60 kB limit. The vulnerability lies in the fact that multiplying large numbers takes a relatively long amount of time. By chaining multiple such multiplications (in this case, `9^5000*9^5000`) into a single request, the runtime of the request may be inflated arbitrarily and increased linearly in the number of multiplications. The multiplications are chained together with alternating additions and subtractions to prevent any of the intermediate values from growing above the limit of 10^50000 implemented in the `BigInteger::BigInteger(int[], int, boolean)` constructor. It is believed that multiplications are slow due to the complex operations performed in `BigInteger::reformulateMultiply()"

Take-Home Evaluation:

The challenge program contains an intended vulnerability in the parsing of input expressions. The potential unintended vulnerability reported by Two Six Labs was confirmed.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.5.3 Question 009 (AC Space, Unintended Vulnerable, Take-Home: Yes)

Take-Home Response:

"[…] A large amount of memory is utilized by the challenge program when it processes input with a large number of characters. Unlike calculator_1 the input to the challenge program is capped at 50,000 characters rather than 10,000 characters. […] Running our exploit against the challenge program gave the following result:

Address	Kbytes	RSS	Dirty	Mode	Mapping
total kB	15755600	4244764	4230180		

which is just over the target 4GB of memory required. The total request takes 49,875 bytes according to tcpdump's length field."

Take-Home Evaluation:

Two Six Labs identified a potential unintended vulnerability that was confirmed.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.6 CaseDB

A.6.7.2.6.1 Question 019 (SC Space, Intended Vulnerable, Take-Home: No)

Take-Home Response:

"There is a side channel vulnerability in space to determine both: [if there is an informant and how many there are]. An attacker can also determine the length of the informant's name, but not the name itself. The results of a GET request are returned in a series of 5 packets to port 6166. The data portion of each packet contains an 8-byte application header and a variable length base64 encoded data block. The base64-decoded data block is broken down into a 16-byte header followed by a variable length data block containing the retrieved documents in this packet: […]

The document string for a public document for caseA with WitnessB is formatted as: caseA:WitnessB. If the document is not public, each character in the document string is replaced with an 'x' (as shown in ReturnRouteImplementer.sendPlan(), lines 184-188, JD-GUI decompiler). Additionally, there can at most be one secret document per case (per SecureDocControlllerX.addDoc) and documents are always returned in the same order between GET requests - the only difference being the addition of a secure document if present, which is also always returned in the same location. In order to determine the number of informants in a GET request (either 0 or 1) and the length of that informant's name, an attacker must do the following

1. Observe the packet sizes of a target's GET request
2. Make an identical GET request from her own unprivileged user account.

If there was no informant in the target GET request, the packet sizes will be identical. If there was an informant, exactly one packet size will differ. The attacker can take the differing packets, remove the 8 byte headers, and calculate the difference of the base64 decoded data lengths (unencoded length = 3 * encoded length / 4 - attacker may have to round down since the app encodes without padding). The difference between the lengths will indicate the x's shown in the target's terminal – the length of the informant's name is this difference minus the length of casename minus 1 (for the colon). There are insufficient oracle queries to brute force the informant's name since the informant's name can be any character upper or lowercase (except for the first character, which must be uppercase), which would require 51*52^(n-1) queries for a name n characters long."

Take-Home Evaluation:

Two Six Labs was very close to the intended vulnerability. Working with Vanderbilt and GrammaTech at the collaborative engagement they should be able to identify the intended vulnerability.

Post-Take-Home Analysis Ruling: Incorrect

A.6.7.2.6.2 Question 021 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“There is no side channel in time for determining the number of informants in a GET request. The timing differences between a privileged target user's GET request and the same GET request made by an unprivileged user is approximately 0.01 seconds - this is regardless of whether or not there is an informant in the response to the GET request or not (and regardless of the length of the informant's name). Even though ReturnRouteImplementer.sendPlan concatenates the informants name to a series of x's using the slower method of "+"-ing Strings together (as opposed to using StringBuilder), this does not appear to cause a significant enough delay between different GET requests on a privileged user's account.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.6.3 Question 036 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“There is a strong timing side-channel in the challenge program, which can be used to get the information that enables an attacker to view secret documents, but it is not achievable within the input budget. […] The issue with the given implementation is that it suffers from a padding oracle side-channel attack. Both the client and the server utilize PKCS5 padding for the password. […]

We have two attacks, first to discover the plaintext password (which is achievable within the input budget) and second to provide a valid ciphertext, IV pair that will pass the GET request for a privileged user (which is not achievable within the input budget, but necessary to answer the challenge question). […] In the worst case this takes 8*39=312 queries. […] Once the intermediate value is determined (this time requiring 8*256=2048 queries), we can select an IV so that XORing the intermediate value with iv returns the plaintext password, allowing us to submit a valid GET request as the privileged user.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.7 Chessmaster

A.6.7.2.7.1 Question 006 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“There is no side-channel which leaks the user's password, so the only attack vector is determining the target user's game state by timing the sequence of moves they send to the server. However, the target user can easily construct a game of unbounded length, thus exhausting the attacker's input budget. […]

Since the server only takes the current state into account when planning its moves, it will make the same move it made when it previously encountered this state. This means that if the benign user repeats these commands they can construct games with an unbounded number of turns. In

addition, each game move is a new POST request to the server, so by the operational definitions observing each move requires 1 passive operation. By constructing a game state with over 30,000 moves, it is impossible for the attacker to discern the benign user's game state after the 30,000th move and so impossible to affirmatively answer the question."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable. We note that the team highlighted an ambiguity in this challenge question. While there exists states where the target user may trigger an un-ending sequence, the intent for this question (and indeed all "game-based" questions) was to treat the benign target user's game move from each state as a uniformly random draw from the set of valid moves at any given game state. We will work to clarify and distinguish this from the concept of the worst-case secret in future challenge questions.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.7.2 Question 015 (AC Time, Intended Vulnerable, Take-Home: Yes)

Take-Home Response:

"the AI is invoked with a call to GameClient.nextMove() to determine the next best move. This method attempts to iterate every possible move for each board piece, and for each possible move, calculate the best move for that particular state until the depth value is reached for successive moves for that piece using a recursive call to GameClient.negamax(). The depth is calculated using the following formula with mostly constants to form a parabola roughly between 2 to infinity. […]

With a high depth value, the Chessmaster server will attempt to calculate a large number of moves and prevent benign requests from getting processed. The only variable that can be user modified in the depth calculation is the variable 'x'. 'x' represents the point differential between the server's max possible score and the current opponent playing the game's max score. Values for each player's score is calculated in State.totalPieceScore() by summing each board piece that exists. […]

We use the promotion mechanism within the game to promote a board piece to another King so that the point differential can be close to 20,000 resulting in a depth of 11. To achieve the King promotion, we leverage the Autocorrecter feature to bypass the condition (lines[4].toLowerCase().equals("king")) used to guard against King promotions by sending the promotion command "b7 b8 \n kiNg" resulting in the string " king" which does not equal "king". The Autocorrector then finds the best match to "king" yielding a depth of 11."

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.8 Class Scheduler

A.6.7.2.8.1 Question 012 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"[…] We performed file uploads with varying numbers of students and teachers, and the "validation" time appears to be constant (or rather, any differences are far too small to detect

while observing packets over the wire). Additionally, when processing the file and creating a schedule, there are too many variables in the input file that can alter the processing time (students, teachers, rooms, courses, max number of generations, etc.), of which we cannot distinguish the values based on schedule processing time alone. For example, even with a constant number of students and teachers, by altering the other variables mentioned, the processing time alters greatly.

Additionally, without login credentials, it appears there are no active operations that can actually be utilized. […]"

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.8.2 Question 017 (AC Time, Unintended Vulnerable, Take-Home: No)

Take-Home Response:

"[…] Because the genetic algorithm performs computationally expensive "mutations" at each iteration to attempt to create a schedule with the least amount of "failures", maximizing this number of iterations appears to be the most promising way to perform an AC attack. By launching this attack multiple times, all cores of the reference platform will be at 100% for the desired 50 minutes. While doing this does exceed the input budget (not by an excessive amount, though), there is likely a smaller school data file that can be used that has the same effect. For reference, the school data file is included. However, similar to EffectsHero question 25, we answer "No" due to the Vaadin framework being used. As a result, most benign requests will be responded to instantly, even with the CPU at 100%, as a new execution thread is created to handle new requests. Hence, it appears that no matter what effect is generated on the ClassScheduler server, it will respond instantly to any benign requests, making the question infeasible to achieve. As the only plausible attack vector appears to be attacking the Vaadin framework itself, which is out of scope, we chose to answer "No"."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

A.6.7.2.8.3 Question 028 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"[…] When the user submits a login attempt, the authentication check is simply using Java's String.equals() method on both the username and password. Given that our active operations are limited to login attempts (there is no functionality that a user can access without logging in first), all we can do is attempt to login to the single hard-coded user. However, the .equals() method does not leak any information about the login attempt via timing, as it is not computationally expensive. Furthermore, there are no prefix-style attacks in time that tell us when a substring of our login attempt was correct (again, it's just .equals()). […]"

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.9 Effects Hero

A.6.7.2.9.1 Question 010 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“Observing the file upload (which is under TLS) does not appear to leak any information about the contents of the file, as there does not appear to be any processing in FileInputBlock that is dependent on file contents. At best, one may be able to obtain an estimate on file size. The same appears to be true for downloading the processed file in FileOutputBlock. Thus, the processing of the uploaded file is the remaining attack vector. Because the target user can apply an arbitrary sequence of processing blocks, it is possible that they could apply none (i.e. only having a FileInputBlock and a FileOutputBlock), in which case from the previous reasoning it would be impossible to determine the contents of the file. However, even with the target user applying valid processing blocks, the combinatorics of the possible combinations (along with the individual options for each processing block) is massive. More concretely, the equalizer processing block has 32 equalizer "bands", each of which can take 200 values. In our testing, it appears that changing these values does result in the expected change (i.e. that of an equalizer) in the output file, but the timing signatures from observing the different values reported to the server are the same. Thus, it appears if a target user is attempting to use the equalizer block, we will have no way of identifying what specific changes are being made to the input file, as the combinatorics far exceeds the operation budget. Even if we were able to identify the changes, without knowledge of the input file, we are unable to determine the correct output file. Thus, this counterexample indicates that there is no side channel in time that will leak the output file of the target user.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.9.2 Question 025 (AC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“While we are able to generate an effect within input budget for this problem (i.e. the CPU of the reference platform is at 100% for 450 seconds), it appears that the Vaadin framework being used for the server will still allow most benign requests to be responded to instantly, as it uses a new execution thread to handle new requests, which still shares CPU time with the thread that generates the effect. Hence, because of Vaadin's ability to instantly respond to new request, we answer "No". For completeness, we describe our attack below anyways, which spins all 4 cores of the reference platform to 100%. In our analysis, we determined that Spline.interpolate() was one of the most expensive method calls. In the BaseBlock class, the fade() method is one of the few methods to call interpolate(). Additionally, the BeatBlock class makes the most calls to fade(). So, we analyzed how to maximize the number of calls to fade() with a BeatBlock processing block, in BeatBlock.process(). We determined that in order to maximize these calls, one must increase the sample duration slider to its maximum value, use a BPM of 330750, beat durations of 1, and alternate check-boxes of a beat matrix. By adding many of these BeatBlocks, we can get CPU usage to 100%. Additional requests will clobber additional cores as well. While

we were unable to script this because of complication (the example script simulates clicking HTML elements in the browser, and the server does not otherwise have an API), performing in a browser is simple. Additionally, our testing of this was below operation budget. Again, we note that while the server was spinning at 100% CPU utilization, any new requests were immediately responded to by the Vaadin framework using a new thread for the request. The sole exception appeared to be when the request came from the same browser that launched the attack (using any other browser gets an immediate response). Hence, the only thing plausible to perform the attack appears to be attacking the Vaadin framework itself, which would be out of scope."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable. The team accurately noted that due to the implemented framework, computationally expensive operations by one user have minimal impact on the request to be delayed.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.10 Railyard

A.6.7.2.10.1 Question 013 (AC Space, Unintended Vulnerable, Take-Home: No)

Take-Home Response:

"No, there is no algorithmic complexity vulnerability in space that would cause the challenge program's memory usage to exceed the resource usage limit given the input budget. Acting within the intended constraints of the program, it is clearly impossible to inflate the resource usage to the degree required by the question within the input budget. All objects manipulated by the program (Cars, Cargo, Bins, etc) are simple and at most linear in size in relation to the size of the user input that produces them, with coefficients on the linear relationship far smaller than the 1000 required by the ratio of Resource Usage Limit to Input Budget. Thus it is impossible to increase the resource usage beyond that limit by creating a train with a large number of cars, cargo, stops, personnel, or some combination of the above. Creating multiple trains has no particular impact on the memory usage of the application, and neither does changing the packing method used by the `BinPacker`. Giving trains or cargo with very long names does increase the RAM usage somewhat, but this increase is again insufficient (at several hundred MB) to exceed the resource usage limit. Further progress may be made by using a variant of the attack used for Question 018. By creating a train with two coal cars (see writeup for Question 018 on how this is to be accomplished) and four cargoes of 1 unit of coal each, each with a 5000 character description, the challenge program will, when instructed to pack and send out the train via a call to `send_out.json`, will infinitely loop in `Platform::printToStream()` over the second coal car and its contents of two coal items with long descriptions. As this produces a file of unbounded size the request will eventually terminate in an Exception when the file consumes all space on the hard drive of the reference platform, causing a call to `stream.write()` to fail. Before that, however, the request will cause the challenge program to consume ~1.3 GB of RAM. This is less than the Resource Usage Limit of 5 GB, though, and so does not constitute an affirmative solution to the question, and so the answer remains no."

Take-Home Evaluation:

Research teams discovered an unintended vulnerability during the collaborative engagement.

Post-Take-Home Analysis Ruling: Incorrect

A.6.7.2.10.2 Question 018 (AC Space, Intended Vulnerable, Take-Home: Yes)

Take-Home Response:

“The railyard manager writes to a single file (out.txt), and occurs when the 'send_out' API call is received through an authenticated HTTPS POST request. Before writing to the out.txt file, all cars added to the train are linked together using the Platform.linkCars() method, where each car's 'next' variable of type 'Object' is assigned a reference to the next car in the LinkedList. We find the Coal Car's 'next' variable interesting because it is a static variable, and if two Coal Car objects exist within the same train, linkCars() will assign the 'next' variable of the first Coal Car and the last Coal Car at the same time. If no cars follow the last Coal Car, linkCars() will not overwrite the 'next' variable, effectively creating a loop of train cars. When cars are added to the train using the Platform.AddCar() method, the railyard manager tracks the cars that have been added using the car type and a user provided car identifier in the format of [<car_type> <car_identifier>]. Attempting to add the same car type and car identifier will fail because it already exists in the manager. Coal Cars are handled differently, their car type is used but the identifier is never concatenated, preventing multiple Coal Cars and Cabooses from being added. We are able to add multiple Coal Cars and bypass checks by adding a "coal car" as the first Coal Car, and "coal_car" as the last Coal Car. Following the LinkedList where train cars are added to, we see that in the Platform.printToStream() method a while loop contains the condition (currentObject != null) where 'currentObject' references the 'next' variable of the car using Java Reflection from the following code: […] Since we have a train of cars linked together where no cars within the train have a 'next' value of null. We are able to remain inside the while loop in Platform.printToStream() and continue writing to the out.txt file until the system runs out of disk space or the railyard manager is terminated.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.10.3 Question 035 (SC Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“No, there is no side channel vulnerability in space from which an unprivileged attacker can discover the cargo materials in the most recently generated railyard schedule. In order to discover the cargo materials in the most recently generated schedule the attacker would have to do one of the following things (note that the contents of the communications between the manager and the server cannot be inspected deeper than the packet level due to TLS encryption – it is impossible for the attacker to measure the length [space] of any subcomponents of requests or responses):

1. determine the contents of the response to the manager's call to `/send_out.json` with sufficient detail to discern which cargoes are listed as being packed onto the sent out train;
2. determine the contents of the manager's calls to the `add_car.json` and `cargo/add.json` endpoints with sufficient detail so as to be able to know which cars and cargo have been added to the train, as this information can be combined with knowledge of the cargo packing algorithm (see the `BinPacker` class) to determine which cargo materials will be packed onto the train;

3. determine the password of the manager and then, after authenticating as the manager with the stolen credentials, call the `last_schedule.json` to obtain a copy of the packed train schematic and thus the cargo materials.

1) is impossible because the manager may give the cargo materials packed on the train arbitrary names of arbitrary lengths and these names are included in the response to `send_out.json`, which effectively adds unknown and indeterminate padding to the response, blinding the attacker to the contents of the response. While the attacker can guess at the lengths of the names (ie padding) by eavesdropping on the manager's requests to `cargo/add.json`, this is an imperfect countermeasure, as the manager may add cargo that is not ultimately packed onto the train, which the attacker cannot determine, by adding cargo and then making calls to `DELETE cargo/:cargo.json`, which leaks only length of the id of the cargo, which is far too weak a signal to determine which cargo was removed.

2) is impossible because the manager may give arbitrary length names to both cars and cargo on adding them to the train, so it is impossible for the attacker to distinguish, from the size of the requests alone, what the type of the car or cargo added in a given request was. (Note that the server's responses `add_car.json` include both the name and type of the car and so present the same problem as the request and its responses to `cargo/add.json` are constant size, differing only based on whether or not the cargo was successfully added). This attack vector is also impossible because the attacker, under the Operational Budget, can only observe 14 requests via Passive Observations, while the manager may make any number of requests, thereby exhausting the ability of the attacker to extract information via side-channel.

3) is impossible because while the manager's request to `authenticate.json` does fuzzily (modulo TLS padding) leak the length of the password, it does not leak any information about the content of the password, and the server's response is constant and thus leaks no information whatsoever about the password. As the password may be of any length, the 14 operations available to the attacker under the Operational Budget are of course woefully insufficient for a brute-force attack on the password via Active Operations. As all three attack vectors are impossible, there is no side channel vulnerability in space."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.11 SimpleVote 3

A.6.7.2.11.1 Question 001 (SC Time/Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"We first note that there is a side channel in space and time which allows a userto determine a target voter's registration key for a ballot. There is a finite set of possible registration keys, and the time it takes to verify a registration keyis determined by the size of the integer formed from a specific 6 characters of thekey. More specifically, RegistrationGate.checkCore calls LogicticMapping.logisticMappingmore times if this integer is larger. An attacker can precompute the times to check eachregistration key, and then compare with the time taken to check the target user'sregistration key when they access their ballot.However, the effect of obtaining this registration key is mitigated by the call tocheckRegistrationCode, which is called following the check for whether you have entereda valid registration key. The

checkRegistrationCode method allows access to a ballot only if the entered registration code has been issued to the logged in voter, for the electionthey are attempting to access. Since the attacker does not have valid credentials to signin as the target user, we believe they will not be able to pass checkRegistrationCode andaccess the target user's ballot. We explored other attack vectors, such as stealing theuser's login credentials, manipulating our own session information, or stealing the user'ssession ID. We believe the registration code itself is insufficient information to view thetarget user's ballot. Our conclusion is that there is no side channel vulnerability in space and time from which an attacker can obtain a ballot belonging to a target voter."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable. The team noted a potential side channel in time vulnerability based on mapping the response time to process a registration key to a given registration key. We believe that there exists a side channel in time to leak half the registration key, but the other half of the key cannot be deterministically mapped. The team reached the correct conclusion in the lack of a vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.11.2 Question 033 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"To obtain a target voter's profile, one must obtain either their password or their session token. The processPassword() method does perform plenty of computation (i.e. encrypts with DES, compresses, etc), however all of that computation appears to be independent of the content of the password. While the DES implementation is custom, it appears to be roughly constant time regardless of the password. At best, an attacker may be able to determine the length of the password rounded up to the nearest 8-byte quantity, given how the number of DES rounds are calculated. Furthermore, in processPassword(), a random boolean is used to determine certain code paths. Lastly, the implementation of Deflate used is the standard Java implementation, which is likely safe from any timing side channels (again with the only possible exception being length). Session tokens are handled much more simply, using a standard Java method for generating UUIDs, and simply setting or retrieving them as cookies. There does not appear to be anything computationally intensive performed in the setting or retrieving of session tokens, and there does not appear to be any measurable UUID-dependent code paths. Thus, without a means to obtain either the password or session token with a timing side-channel, we have no means to obtain the voter profile. Furthermore, timing measurements of various password attempts appear to be identical."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.11.3 Question 034 (SC Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"The only mechanism by which percentages of results might be leaked is during results aggregation when precincts communicate their vote totals. Editing a free response question on any ballot, beginning the response with a space, appending some number of spaces to the end of the response, and saving the ballot will update the value of

ResultsMessageCreator.INCLUSION_THRESHOLD to the number of appended spaces. This happens as a result of calls to udpate1 and updatel in EditBallotManager.legitimizeChoices. The threshold value is used as a lower bound for which candidates shoud be included in the SummaryTally. This determination is made by ResultsMessageCreator.shouldInclude. However, the comparison in shouldInclude is `count / totalCount >= INCLUSION_THRESHOLD`. The count value will never be greater than totalCount, so `count / totalCount` will be a value between 0 and 1. INCLUSION_THRESHOLD, on the other hand, is an integer. The only introspection an attacker could hope to gain is setting INCLUSION_THRESHOLD to 1 and potentially learning which candidates, if any, have earned 100% of the vote in their election. However, this is insufficient information to determine the percentage of finalized votes for any target candidate. We note that brute forcing the percentage using 75 oracle queries would allow an attacker to succeed 75% of the time, however there is no way to increase this above the require 90% probability of success.Our conclusion is that there is no side channel vulnerability in space from which an attacker can determine the percentage of finalized votes received by a target candidate in their best precinct to within 0.5 percentage points."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.11.4 Question 037 (SC Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

"[Similar] to question 33, the only means to obtain the target voter's profile is to either obtain their password or session token. All session tokens are randomly generated UUIDs, and are not distinguishable by size, hence obtaining the target's password is our only option. The processPassword() method performs compression on a string containing both the correct and submitted (encrypted) passwords. The length of the compressed string is returned to the user on a login request. However, when constructing the string to be compressed, a random subset of characters from the submitted password is used, because of a call to random.nextBoolean() in processPassword(). This has an effect on the length of the compressed string, as it is no longer deterministic, and has a wide range of possible values (it should be noted that in a previous version of simplevote, this length was utilized to determine the password by guessing character-by-character and looking for the smallest compressed length). Furthermore, in testing the distribution of lengths for various passwords, there does not appear to be any pattern or consistency that could be exploited. Hence, there is no side channel in space that leaks information about the password, and whether any prefix (or more generally, subset) of our guessed password is correct. Thus, without any side channel in space for either the session token or password processing, we determine that we are unable to determine a target voter's profile."

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.12 SimpleVote 4

A.6.7.2.12.1 Question 030 (AC Time, Intended Vulnerable, Take-Home: Yes)

Take-Home Response:

“There is an algorithmic complexity vulnerability in time. The ChaoticOperator.chaoticOperator method calls Stash.considerAdding. This function takes a very long time to return if isCrowded() evaluates to True. This can occur when this.content - 1 is entered as a value in the Stash. In ChaoticOperator we can see this.content is initialized to 996,so we need to add item 995. At first, 995 fails the SanctionAuthenticator.isInBounds check because the input must be less than MAX_N, which is 995. However this method has the side effect of increasing MAX_N if the input is equal to MAX_N, so if we submit a registration key with nStr 995 twice in a row, then 995 is added to the Stash. Submitting the registration key a third time triggers the Stash.composeOpening function in Stash.considerAdding, and any benign requests at this point will be delayed for more than the requisite 300 seconds. There is an exploit.py script provided which launches this attack. The PDU size of application requests sent to the server is 3096 bytes.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.12.2 Question 039 (SC Space, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“As a voter in precinct A we can edit our ballot. If we as the attacker edits a text field such that our input:

1. Begins with two spaces.
2. Contains up to 20 characters of "data" (more on this in a moment)
3. Contains an appended amount of spaces equal to the desired threshold

Then the effect on the summary string and the associated Ternary Search Tree (TST) is as follows:

1. Candidates are included in a summary string (sumStr) if the percentage ofvotes they received was above the given threshold. More on this below.
2. The data characters are appended to the end of the sumStr,and are used in the generation of the TST.

The sumStr is constructed to include the names of only the candidates who have recieved a percentage of the vote that is above a particular threshold. This is done in ResultsMessageBuilder.shouldInclude, which checks whether the percentage of votes for a candidate is above INCLUSION_THRESHOLD. The INCLUSION_THRESHOLD can be manipulated to be any whole number following the steps to edit a ballot described above. This is a result of calls to update1 and updatel in EditBallotGuide.legitimizeSelections.

The construction of the TST is unconventional. Rather than adding whole words to the tree, strings are added in the following way:

1. First each of the 256 bytes are added as a node.
2. For each substring of the sumStr, the longest prefix is foundin the TST. This series of nodes is then extended verticallyby the next character following the longest prefix in sumStr.
3. The final character of each added substring is then used as thefirst character of the next round of processing. For example,'Tree' adds T-r then r-e then e-e to the tree.

4. If the sumStr is long, this process is cut short once 2048 nodes have been added to the TST.

After the tree is created, the sumStr is processed by the tree. This is achieved by finding the longest prefix of the sumStr, printing out the node value (the key for that node) and then moving to the next portion of the sumStr. Each node in the TST accounts for 20 bytes on the wire, and each node value in the parsing accounts for 4 bytes. Therefore, there is a space side-channel in both the TST size and the number of nodes. Our attack works by generating two TSTs of equivalent size, but where one will cause fewer node values to be printed because a candidate's name is parsed in the generated TST more efficiently. Below we give two examples of inputs used in our messages. For this example, Hillary Clinton is the target candidate, but any candidate will do (modulo some issues we will point out later).

Benign: *ZY@ZYX`ZYXW$ZYXWV

Candidate: *Hi@Hil`Hill$Hilla

Because of how the TST is generated, the lower string will add the path H-i-l-l-a to the tree, however the effect on tree size of both of these strings is equivalent. Note it is very important that the "delimeter" characters we chose (e.g. *, @, etc.) do not appear anywhere in the sumstr. This means that the TST will build the way we expect, adding *-Z, Z-Y, Y-@, @-Z, Z-Y-X, etc. As long as the benign string never appears in the sumstr (which again we can tailor since we know the ballot), the two generated TSTs will have the exact same size. This means that if Hillary Clinton has over the supplied threshold percentage of votes in our precinct, she will be included in the sumStr and the resulting message for the Candidate input will be shorter than for the Benign input. These messages can easily be observed as the servers use designated ports solely for this aggregation task.

Note: there are some challenges if two candidates share the same prefix in their name, but our attack can easily be adapted to handle these corner cases because we have knowledge of the ballot (we can select other portions of their names).

Note: As mentioned above, the TST has a max size of 2048 nodes. If the threshold is too low, there can be many candidates above the threshold. If this happens, not all of the sumStr is included in the TST and our attack will fail. However, we note that threshold imposes a limiton the number of candidates included in sumStr (e.g. at most 3 candidates per question can have a least 30% of the vote), so we do not encounter this issue unless the target candidate recived a small percntage of the vote. We believe our operational budget is sufficiently large that additional oracle queries would allow us to resolve any uncertainty from this issue. By changing the threshold value, we can binary search onto the percentage of votes for the chosen candidate in our precinct. We can also send our TST and threshold to the server for precinct B, and they will parse their sumStr using the TST. However, at this stageserver B computes the aggregate results for precincts A and B, notthe results for precinct B alone.

The attacker can similarly perform binary search to recover the aggregate percentage for the target candidate based off responses from server B. However, this does not leak the percentage vote for the candidate inprecinct B. If precinct A has fewer than 15 percent voting for the candidate in question and the aggreggate is larger, but still lower than 15%, then there are configurations of numbers of voters in precincts A and B that make it infeasible to guess the percentage in precinct B using oracle queries. For example, if precinct A has a much larger voting population than B, then B could potentially have any percentage between the aggregate

percentage and 100%, simply because the total number of voters in B will barely affect the aggregate percent. In this case, despite knowing the candidate's total percentage of votes, and the candidate's percentage of votes received in precinct A, we will be unable to determine the percentage of votes received by the target candidate in their best precinct (precinct B) to within 0.5 percentage points within the operational budget. We also did not identify any side channel which would leak the absolute or relative numbers of finalized votes in each precinct.Because the problem description states that we only have voting credentials in precinct A, and we cannot necessarily determine the exact voting percentage in precinct B in the worst case, the answer is "no".”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable. The team identified an interesting vulnerability by which an attacker can potentially determine the secret in the precinct where the attacker is registered to vote. While this has no impact on the evaluation of the response, we believe that in the worst case, even at a threshold of 33 (and possibly even higher), there can be enough data in the summary string that the seed won't affect the trie. So, assuming it's even possible to determine whether the percentage is less than 34% using the SC (with 10 operations, plus login), one would need up to an additional 68 oracle queries to narrow down the percentage sufficiently.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.12.3 Question 041 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“There is no side channel in time from which an attacker can gain knowledge of a user's profile. Since we are only guaranteed to observe a user logging in, we started our analysis at LoginGuide.handleDelivery(),the function which handles the GET login request. We identified two attack vectors: the password comparison and the generation of the session cookie as either of these pieces of information can be used to obtain the user's profile (either by logging in with the stolen password or by by-passing login by sending a GET request to /profile with the session cookie). The password comparison runs in proportion to the length of min(actual_password, supplied_password). An attackercould potentially leak the length of the password by sending passwords of increasing length and observing theresponse time of the server - the response times should increase as the length of the guessed password slowlyapproaches that of the actual password, and then be relatively constant once the guessed passwords passed the length of the actual password. However, the differences in timing are not likely to be observed over the wire (if at all, given the number of instructions executed while logging in) and the data obtained is insufficient to brute force the password give the operational budget. The second attack vector, the cookie generation, does not appear to leak any information as it uses the standardJava method UUID.randomUUID() to generate the cookie returned to the user. This cookie is encrypted when it is sentback to the user over HTTPS. Finally, we turned our attention to the actual data we are trying to leak: the voter profile. We could not find any place in the code where this information is accessed and could leak information by observing network traffic timings.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.13 Swappigans

A.6.7.2.13.1 Question 002 (AC Time, Intended Vulnerable, Take-Home: Yes)

Take-Home Response:

“Attempting a purchase triggers a large amount of recursion, as the server will seeks to find an optimal trade for the item, via the apxSumRecursion method. By having a large amount of carefully priced items (which the attacker can add at-will), this can cause apxSumRecursion to maintain many possible trades with each recursive call, which is quite expensive. An exploit script is provided that will add the needed items, request to make a specific purchase, and simulate a benign user attempting to login, with timing information provided. It should be noted that the list of items that are added in "prep_exploit.py" are by no means optimal, and there is likely a set of items to add that is smaller and still achieves the desired effect. That being said, the current exploit script consistently achieves above 6 seconds with 34.3 kB sent.”

Take-Home Evaluation:

As with several other teams, Two Six Labs did not identify the intended vulnerability that requires exceeding the $1000 price limit. Instead, the team used the fact that the number is part of the complexity function to exceed the resource usage limit.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.13.2 Question 020 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes)

Take-Home Response:

“The getSaltString() method is called on the first login or registration since the server has started, being a static method in the SessionTokenManager class. It seeds a java.util.Random object with the current time in milliseconds. The "Random" object is used to generate a constant AES key that is used for all future logins and registrations, by encrypting the username to create the session key. Importantly, we can assume that no other users are active on the server, and from a clarification email, we are able to assume that the target user sign-in is the first activity on the server since its start. Hence, when we observe the timing of the target user login, that time will be very close to the time that was used to seed the "Random" object, which will have occurred just before the network packet was sent/seen. Hence, the attacker can then begin the following process: while brute forcing over possible times in milliseconds (going back from the observed time a maximum of approximately 2 seconds, so as to stay within operational budget), seed "Random" with that time and create the AES key exactly as it is in getSaltString(). Then, use that key to encrypt the username. Finally, perform an oracle query on if the resulting session token is correct. If it is, the attacker can successfully login as the target user and make a purchase with their items. This approach requires some accuracy of network timing (about two seconds), which shouldn't be an issue. Furthermore, because the current time is divided by 1000 before being set as the seed, the brute force actually only needs to be over seconds, rather than milliseconds. Hence, one passive operation is used to observe the target user sign in, and only a few are needed for oracle queries (say 2 or 3). A simple brute force program was created to successfully confirm this vulnerability, and is included.”

Take-Home Evaluation:

Two Six Labs flagged the same unintended out-of-scope vulnerability as several other teams. A key assumption of this vulnerability is the attacker knowing when the server was started. In

future challenges we will work to clarify that the attacker does not know when the server was started.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.13.3 Question 031 (SC Time, Intended Not Vulnerable, Take-Home: No)

Take-Home Response:

“While we observe the login time, this won't provide any information on the item that the target user requested to purchase. While observing the purchase calculation time does provide information, it is highly dependent on what items a user currently owns (of course, in addition to the requested item). Without knowledge of what items a user currently owns, this appears to be unfeasible with 32 operations. For example, we obtained timing data for a user with one item who made various purchase requests - all appear to have very similar timing signatures. However, there is an attack vector to determine the items a user owns via the exploit of Question 20 - i.e. using the timing information of when the user signed-in/registered to obtain their session key. Because purchase requests in Swappigans don't actually alter the items a user owns (just calculates which items would need to be traded), if we perform this exploit after the user performs the "purchase", we will still only see their items before any trading has occurred (the purchased item never actually enters the user's list of items). Unfortunately, this does [not] appear to help - at best, by obtaining the user's list of items, we could make all purchase attempts with those items offline using an attacker-controlled server, and store their timings. Then, we could use oracle queries on the purchase requests that correspond to the closest timings. However, as this attack would need to work 85% of the time for any possible list of items that a user owns, this appears infeasible. For example, as previously discussed, a user who has a single item has near identical timings for most purchase requests, so reproducing all timings with that item offline does not provide much information.”

Take-Home Evaluation:

Two Six Labs correctly concluded that the challenge program is not vulnerable.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.14 Tollbooth

A.6.7.2.14.1 Question 011 (AC Time, Intended Vulnerable, Take-Home: Yes)

Take-Home Response:

“The TollBooth manager accepts incoming 'new_node' packets from new leaf peers. The data within a new node packet is not encrypted and contain different types of information that is serialized using the PacketPayload class. Payloads containing literal strings are serialized by the method putStringIntoByteBuffer() and deserialized by the method getStringFromByteBuffer(). During deserialization, the length of the string is parsed, followed by an allocation to a new buffer using the parsed length as the size, and then completed by copying the string from the payload into the allocated buffer. Payload deserialization for strings: final int len = buff.getInt(); final byte[] data = new byte[len]; We can maliciously send a new node packet with the incorrect string length to the root manager causing a java.lang.OutOfMemoryError exception. Once the exception occurs further benign charge_transponder requests are not processed.”

Take-Home Evaluation:

Two Six Labs identified the intended vulnerability.

Post-Take-Home Analysis Ruling: Correct

A.6.7.2.14.2 Question 024 (SC Time/Space, Intended Not Vulnerable, Take-Home: Yes)

Take-Home Response:

“We examined the network traffic between the root manager and a benign leaf peer where two different types of packets were observed using tcpdump; 'CarPassedPacket' and 'TransponderChargedPacket'. The type of the packet is displayed conveniently at offset 0x7A followed by the transponder ID at offset 0x93 for Transponder Charged packets and offset 0x8A for Car Passed Packets. We display the Transponder Charged packet as reference below […] Information regarding a transponder's password or session token from the root manager's Web server are not found within the observable packets or cannot be deduced from the timing of packets sent or received. From the example above, we are able to determine the transponder ID and other payload information that we will skip for this explanation. In the code snippet below, we have found that an attacker can transfer funds from a target user's transponder using only their transponder ID and the web server's transfer funds feature. In the do_add_value() method, an attacker can completely bypass the authentication mechanism by supplying the variable 'token' as an array of size greater than 1. The value of 'paramValue.size()' is checked on line 2 for a value of 1, with a size greater than 1, the entire 'if' block is not taken and the token is never compared with the user's password (line 5). […] The source and recipient transponder ID's are supplied using an HTTP POST request as the references 'transponder' and 'transfer_id' respectively. We can swap out our transponder ID as the source of the funds with a target's transponder ID and declare our transponder as the recipient. The method TollBoothRootWorker.transferBalance() will [perform] the transaction given adequate funding of the source without any further authentication. Thus, we can arbitrarily submit transfer requests of any transponder ID and if that transponder exists with funds, we will successfully transfer the funds to our account without knowing the target's password or token. We have a proof of concept script which transfers funds from ID 702 to ID 1494 without knowing 702's credentials. The script can be altered for any attacker or victim ID. The script siphons 2 funds from the victim to the attacker's balance.”

Take-Home Evaluation:

The challenge does not contain an intended vulnerability. Two Six Labs identified an unintended out-of-scope vulnerability to leak the secret.

Post-Take-Home Analysis Ruling: Correct

A.7 Engagement 7

A.7.1 Engagement Participants

Included for reference are abridged versions of the teams’ responses to each question and Apogee Research’s evaluation and correctness ruling.

A.7.1.1 UC Boulder

A.7.1.1.1 Answer Statistics

Table A-143: Engagement 7 UC Boulder Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	11	6	88
SC Time	13	12	92
SC Space/ Time	7	5	71
AC in Space	18	17	94
AC in Time	11	11	100
Total	**60**	**51**	**85**

Table A-144: Engagement 7 UC Boulder Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	31	23	74
Not Vulnerable	29	28	97
Yes Answer	24	23	96
No Answer	36	28	78

A.7.1.1.2 UC Boulder Specific Responses

A.7.1.1.2.1 Calculator 3

A.7.1.1.2.1.1 Question 023 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"Yes, there is an algorithmic complexity vulnerability in space. The idea is to have the application compute a number as large as possible with a minimum size of input. Inside the application, numbers are stored in arrays and hence might consume lots of memory if they are large. The exploit input is to send the following query to the functionality of basic calculator for 72 times at the same time:

99518∗99518∗99518∗99518∗99518∗99500∗99500∗99500∗99500∗99478

It is under the budget of 60KB to send 72 requests"

Evaluation:

Exploit did not confirm vuln. From CP: "Colorado's relied on multiplying powers of 99. While their exploit can sometimes exceed the resource bound in the reference environment, the probability of success is approximately 24%, rather than the required 99% (based on 25 runs). We tried tweaking the exploit, but it didn't seem to help. While all the other collaborative teams mentioned Grammatech's exploit, Colorado only mentioned their own, so their answer seems to be incorrect."

Post-Engagement Analysis Ruling: Correct through collaboration

A.7.1.1.2.1.2 Question 025 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"No, there is no algorithmic complexity vulnerability in space. The log file will be written only when exceptions of certain types are thrown. In this program, all exceptions thrown are properly handled, except a java.lang.StackOverflowError exception. AStackOverflowError exception can be triggered by sending an expression with nested parentheses, resulting in writing to the log file. The reason for a StackOverflowError exception to be caught and written to the log file is that, java.lang.StackOverflowError is not an instance of java.lang.RuntimeException. As a result, method AbstractHttpGuide.doHandle will throw java.lang.StackOverflowError and this exception will be encapsulated as oracle.as.scheduler.ExecutionErrorException (according to the API document), hence getting caught in method AbstractHttpGuide.handle"

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.1.3 Question 034 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there exists an algorithmic complexity vulnerability in time. The solutions directory contains the exploit. Iowa confirmed that the exploit delays the response of the benign request by121 seconds. The CodeHawk loop summary points to the LargeNumeral.subtract(see Figure 3) The exploit was found using our fuzzer, the Singularity tool. The number of generations and fitness functions are shown in Figure 1. First we computed a pattern of size 2000 and afterwards use that pattern to create a pattern of 10000. We first started with a pattern of 1000, but a pattern of this length could not be extended, which is the reason we had to go to 2000. The nested loops in this method, however, are all due to Math.random, which in most cases do not significantly add to the time. In this case, however, the loop bound can be up to 10,000 combined with the 3-deep nesting level, an estimated 2 ms is added to each subtract call (based on profiling information and running tests on the isolated code). Profiling confirms that much of the time is indeed spent in the subtract method, however the Math.random method does not appear in the profiling log at all. This could be due to Math.random being inlined by the hotspot compiler. The large number of subtract calls can be explained by looking at the reverse call graph for subtract(see Fig??), which shows that division is performed via subtraction."

Evaluation:

Confirmed their unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.2 Calculator 4

A.7.1.1.2.2.1 Question 036 (AC Space (Memory), Intended Vulnerable, Answered: Yes)

Response:

"Yes, there is an algorithmic complexity vulnerability in space. The idea is to have the application compute a number as large as possible with a minimum size of input. Inside the application, numbers are stored in arrays and hence might consume lots of memory if they are large. The exploit input is to send the following query to the functionality of basic calculator for 60 times at the same time:

999^351∗999^351∗999^351∗999^351∗999^351∗999^333∗999^333∗999^333∗999^333∗999^246∗99

Regarding budget, each request is of size 86+198+554=838 bytes. Hence it is under the budget of 60KB to send 60 requests."

Evaluation:

Colorado's exploit can sometimes exceed the resource bound in the reference environment, the probability of success is approximately 36%, rather than the required 99% (based on 25 runs). Credit is given assuming they worked collaboratively with the other teams but may have not had time to update their answer.

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.2.2 Question 043 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"We believe there is no algorithmic complexity vulnerability. We also applied our fuzzer, Singularity, to Calculator4, but observed that the worst-case time that could be achieved was 6 secs. The performance output from Singularity (see Figure 4) shows the much lower performance compared to Calculator3.

Another difference observed between Calculator3 and Calculator4 is that the corresponding subtract method (BigNumber.subtract does not have the nested loops controlled by Math.random(see Fig-ure??). The time observed for the exploit of Calculator3 on Calculator4 was just 6 secs (20 times lessthan on Calculator3); we do not believe that this large a difference can be contributed to this difference only, but so far we have not found other differences in the code that could explain the difference in running time."

Evaluation

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.2.3 Question 045 (AC Space, Intended Vulnerable, Answered: No)

Response:

"No, there is no algorithmic complexity vulnerability in space. The reasoning and argument are the same as Question 25."

Evaluation:

Incorrect; missed the intended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.1.2.3 CyberWallet 1

A.7.1.1.2.3.1 Question 006 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"No, there is no algorithmic complexity vulnerability in space. The webserver assigns the allow KeyFile to null during initialization, which prevents the program's memory usage to grow significantly."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.3.2 Question 035 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"Yes there is a side channel vulnerability in space. Balances of the users determine their tiers. In ArchiveInfoManager class, the balance determines an image to load from images folder (select an image based on the codes in BankerAssistant class). Since the images have different sizes, this can contribute the space side channel observations. All tiers with Bronze goes to one of images ad00,ad10,ad20,ad30,ad40 (similar sizes around 1.9M), Silver goes to imagesad01,ad11,ad21,ad31,ad41 (similar sizes around 2.1M). Program adds noise from gaussian(0,1). To be 95% confident, the values of secrets (means) should be far by at least 2*Standard Deviation. However, the secrets are far by 10, 100, 1000, and so on."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.3.3 Question 051 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

*Check GrammaTech's answer

Evaluation:

GrammaTech's answer correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.4 CyberWallet 2

A.7.1.1.2.4.1 Question 024 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"A vulnerability similar to the SC in cyberwallet1 in AccountDisclosureManager class. However, in this case, the randomness should not allow to determine the secret with 95% confident. Unlike the version1 where the random sample from gaussian(0.0,1.0) only adds to the secret, in thisversion2, the random sample from gaussian(0.0,1.0) multiplies to the secret value. As a result, the secret value becomes a gaussian random variable with mean 0 * secret and variancesecret2∗1.0. This means that the secret values should be 2∗secret2far from

each other to be observable with 95% confident. In this case, secret values (in the location of becoming random variable) are 10, 100, 1000, ... So, they are not distinguishable with 95% confident."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.4.2 Question 059 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

Confirm GrammaTech's Response

Evaluation:

GrammaTech's response correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.5 Door Master

A.7.1.1.2.5.1 Question 029 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"No, there is not an algorithmic complexity vulnerability in time. During the preparation we expected that there was no vulnerability, but we we not sure enough to answer the question. During the live engagement, GrammaTech pointed us to a potentially endless loop that could be vulnerable. The suspicious loop is in theKeyFobController.authKeyFobprocedure:String result = KeyFobService.Auth(encKey); while (result == null) {result = KeyFobService.Auth(encKey);}It was not clear 1) whether one could trigger the endless loop by having KeyFobService.Auth(encKey) return null indefinitely and 2) if triggering the endless loop will delay the benign request. The co-operation revealed that the answer to 1) isyes: using and deleting a key at the same time causes a race condition that in 50% of the cases causes one thread to end up in this endless loop. However, the answer for 2) is no: the above described procedure only blocks one thread. The budget of 2kb allows this procedure to be performed only once (maybe twice) which is not enough to block all threads."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.5.2 Question 041 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there is a side channel vulnerability in time. The side channel allows the attacker to determine a benign user's key fob private key. In the main loop of the application, there is a ping issued every loop. Every iteration of the ping is delayed by a Thread.sleep()command. Before any key fobs are registered, the delay is 1 second. After a key fob is registered, the delay is equal to the return of a call toKeyFobService.nextInt().The return value for this function is based on the private key of any key fobs. This can be seen by tracking how the return

value is generated. The return value ofKeyFobService.nextInt()is equalto(SecureRandom.generateSeed()[] +1) * 100.SecureRadomis a implementation of the class and generateSeed()is implemented using engineGenerateSeed(). Analysis of engineGenerateSeed()shows that the output of this function is this.temp[count]. The call toMatrixRandomizer.Matrix()is an obfuscated function that does nothing in 99% of runs of the application; in 1% of runs it causes the side channel to fail. The this.temp variable is initialized to zero, and every call to engineSetSeed(arg) appends the argument to this.temp. This is called when a keyfob is constructed on the binary encoding of the private key. This analysis shows that the binary encoding of each key is stored in a secret string in SecureRandom and every call to KeyFobService.nextInt() returns the next bit of the secret. Therefore, the timings between each ping in the system correspond to 1 bit of the private key. The private key is roughly 345 bytes long (plus or minus a couple of bytes). So the attacker can use2760 (= 8) operations to observe the its bit in the private key, and determine the binary encoding of the private key registered by the benign user. This information is looped repeatedly and allows the attacker to determine the length of the key. In worst case this could double the number of operations (so around 5520 operations). The attacker can see where the key starts as a large package can be observed just before the ping requests start that reveal the bits of the key. Alternatively, we also observed that consecutive keys that are generated by the server have overlapping initial bytes. The attacker can read a loop of bytes of the private key, request their own private key to determine the overlapping bytes, and use the overlapping bytes to determine the start of the private key detected from the side channel. The attacker can determine the private key of a user on the server using by observing the timings of roughly 5520 packets, within the budget of 15,000 operations. This allows the attacker to use the observed private key to gain entry using another users key fob."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.6 EMU 6502

A.7.1.1.2.6.1 Question 004 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"Based on discussions with Iowa we believe there is no vulnerability."

Evaluation:

Correct but minimal justification.

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.6.2 Question 012 (SC Space, Unintended Vulnerable, Answered: Yes)

Response:

"Yes, there is side-channel in the space that leaks the source code selected and executed by a user. Note that every operation in the application sends over a unique TCP connection. Every source code has a unique traffic pattern observable through network analysis within 20 first operations. The attacker first profiles every source code on his/her local machine and builds a fingerprint for every source code. On the attack mode, the attacker observes the network traffics

passively and match the on-line observation to one of his/her local observations. Since each file has a unique fingerprint of network traffic, the attacker can reliably determine the source code selected and executed by a benign user"

Evaluation:

Justification correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.6.3 Question 014 (SC Time, Intended Vulnerable, Answered: No)

Response:

"No, there is no side-channel in time that leaks the source code selected and executed by a user. First, we observe the use ofBranchingExceptionin classEmulatorof packagecom.bbn.emulator.In the experiments, we find that it takes about 50 milli-seconds to handle the exceptions. The timing finger prints of each source code can be easily distinguished based on the location and the number of branches in the source code. Note that users cant change the source code of the application. We confirm this by observing that there is no possibility for dynamic invocations. However, within the given budget of 25 operations that only allow to observe at most 25 unique TCP connections, we were not able to find a unique map from our on-line observations to our off-line observations."

Evaluation:

Justification correct given the clarification

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.6.4 Question 016 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Based on discussions with Iowa we believe there is no vulnerability."

Evaluation:

Correct but minimal justification.

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.7 InAndOut 1

A.7.1.1.2.7.1 Question 007 (AC Time, Unintended Vulnerable, Answered: Yes)

Response:

"Yes, there is an algorithmic complexity in time. The vulnerability is that we can send a validation code that is not in use but "appears" to be in use, allowing the program to satisfy the path conditions and then enter a while loop that contains a sleep() method. The cause of this vulnerability is that, this program is intended to have a validation code 1-1 correspond to a binary code. However, in fact two validation code may correspond to a same binary code. CodeHawk tool reports in method com.cyberpointllc.stac.inandout.manager.InAndOutAid.pickupOrder the while loop that contains a sleep method."

Evaluation:

Confirmed unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.7.2 Question 027 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there is an algorithmic complexity in space. The final field COMPONENTSOVENSWITCH in class com.cyberpointllc.stac.inandout.OrderDirector has a value of 10412, which will eventually affect the number of times executing the loop body in methodinandout.CookingCONTROLLER.calculateStageSeries.Furthermore, since this while loop adds an element to a vector in each loop iteration, the number of loop iterations will affect the size of the vector and hence also memory usage. In the exploit, the pie toppings consist of Tomato Sauce, Pepperoni, Onions, Extra Cheese and Pineapple, and the number of quantity is 7. The exploit is confirmed on the reference machine. As a side note, the large number of loop iterations (i.e. 134217727) is not significant enough to cause an AC Time vulnerability. CodeHawk reports in method stac.inandout.inandout.Cooking CONTROLLER.calculateStageSeries a while loop whose loop exit condition is nontrivial, and contains a java.util.vector.add().

Evaluation:

Confirmed close enough to intended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.7.3 Question 049 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there is a side channel vulnerability in time. The value of a binary code that is generated for a pizza order directly affects the response time of the server when responding to the inventory manager. There is an interesting loop in method OrderTaker.obtainIndexFromCONTROLLER whose number of iterations is exactly the sum of values of each number in the validation code (we will elaborate the relation between a binary code and a validation code in the next paragraph). The number of times executing the above loop directly affects the time difference between the second and third network packets ex-changed between the server and the inventory manager. More specifically, the second request is sent from the inventory manager to the server and thus the server will construct a response by invoking method OrderTaker.calculateValidationCode, which invokes methodOrderTaker.obtainIndexFromCONTROLLER. The relation between a binary code (composed of 6 digit) and a validation code (composed of 6 numbers)is that, a digit in the binary code is mapped to a number (which is a power of 2) in the validation code in the same position. The values of numbers in the validation code are determined by the positions of the corresponding digits in the binary code. For example, binary code 0−1−1−0−1−0 is mapped to validation code 0−2^1−2^2−0−2^4−0. As another example, binary code 0−1−1−0−1−1 ismapped to validation code 0−2^1−2^2−0−2^4−2^5."

Evaluation:

Confirm

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.7.4 Question 055 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"No, there does not exist a side channel vulnerability in space. This program exhibits constant network packet sizes for different inputs."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.8 InAndOut 2

A.7.1.1.2.8.1 Question 028 (SC Time/Space, Intended Vulnerable, Answered: No)

Response:

"No, there does not exist a side channel vulnerability in space and time. On one hand, there is no time correlation among different inputs. On the other hand, although there exists space correlation among different inputs, exploiting this does not fit into the input budget."

Evaluation:

Incorrect. No exploit provided (that may have gone over the budget)

Post-Engagement Analysis Ruling: Incorrect

A.7.1.1.2.8.2 Question 039 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there exists an algorithmic complexity vulnerability in time. The definitive vulnerability is the same as mentioned in Question 7. Sending a validation code that is not in use but "appears" to be in use will cause the program to enter a while loop that executes method sleep in each iteration. We also find another vulnerability that is caused by the fact that pizza orders are processed based on their priority values. If we order multiple pizzas, we can force the first pizza order to have the lowest priority, making it the last order to be processed. More specifically, all pizza orders are stored into a priority queue, which is constantly polled from a thread to be processed. As a result, the priority of an order will affect how long it needs to wait before getting processed, which directly affects the response time of this request. The overhead for sending one request is 517+126=643 bytes. The size of actual contents varies from1045 to 2412 bytes. Hence we can send at least 10 requests and at most 19 requests. It typically takes around 5 seconds to respond to one request. To exceed the time limit of 75 seconds, we only need to send 18 requests, and make the priority value of the first order very large in order to get processed at last (i.e. the lowest priority)."

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.8.3 Question 053 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"No, there is no algorithmic complexity vulnerability in space. When the value of COMPONENTSOVENSWITCH in class com.cyberpointllc.stac.inandout.OrderDirector is 2184, the number of times executing the loop body in method inandout.CookingCONTROLLER.calculateStageSeries remains a constant (i.e. 524287). As a result, the number of times executing java.util.Vector.add remains a constant and is not sufficient to launch an attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.9 Litmedia 1

A.7.1.1.2.9.1 Question 009 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)
Response:

"Yes, there is an algorithmic complexity vulnerability in space that would cause the challenge program's memory usage to exceed 600 MB limits given the input budget of 10kB. The string builder DS in the publishingViewerGuider calls handleGrab that builds a html request with tag to load a specific images. IconGuideclass calls handleGet method that reads the images and cause difference images with different sizes are written with stream method."

Evaluation:

Confirmed their exploit for unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.9.2 Question 032 (SC Time, Intended Not Vulnerable, Answered: No)
Response:

"No, there is no side channel vulnerability in time from which an attacker with a user credential can discover the current top-ranked stored preference of another user (target user). Since we did not observe the timing differences, we can only use query operations. There are 12 possible categories and 25 tags. Each doc has one category and up to 25 tags. However, the size of data base is fix to 65unique pairs of (category, tags). So, one can ask up to 65 queries to figure out the current pair. However, the cost of query operation is 10 for this question, makes the total cost 650 without timing observations."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.9.3 Question 056 (SC Space/Time, Intended Not Vulnerable, Answered: No)
Response:

"No, there is no side-channel vulnerability in space and time from which an attacker with a user credential can discover another user's bookmark list. There are 65 different docs. Each user can have up to 42bookmarks. The worst case is choosing 32 out of 65 which requires many observations (65! / (32! *33!). There is clear side-channel in space: the. packet sizes of the bookmark pages differs for users. The dynamic analysis of other teams show that there is

a large number of secret leakage. However, the leakage seems not be large enough to recover current bookmarks of a user as the buckets are too large(given the 200 operations). We did not observe a side-channel in time, which could have provided more leakage. Even with a clear side-channel in time we expect that the answer is no."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.10 Litmedia 2

A.7.1.1.2.10.1 Question 001 (SC Time, Intended Not Vulnerable, Answered: No)
Response:

"No, there is no side channel vulnerability in time from which an attacker with a user credential can discover the current top-ranked stored preference of another user (target user). Since we did not observe the timing differences, we can only use query operations. There are 12 possible categories and 25 tags. Each doc has one category and up to 25 tags. However, the size of data base is fix to 65unique pairs of (category, tags). So, one can ask up to 65 queries to figure out the current pair. However, the cost of query operation is 10 for this question, makes the total cost 650 without timing observations."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.10.2 Question 050 (SC Time, Intended Not Vulnerable, Answered: No)
Response:

"No, there is no side channel vulnerability in time from which an attacker with a user credential can discover the media ID of the target literature by the target user. The cost of oracle query is 10. Before any timing observations, the secret can be any of 65 possibilities. With assumption about uniform distribution over the secret values, there is about 6 bits of information that can be leaked from the secret. After the timing observations, 5.6 bits of information remained in secret bits. This is equal to about 22 possible secret values that need to recover from the oracle queries. This costs in 220 operations that are obviously beyond the allowed budget of 80. Only 7 secret values can be recovered from the oracle queries."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.10.3 Question 058 (SC Space/Time, Intended Not Vulnerable, Answered: No)
Response:

"No, there is no side-channel vulnerability in space and time from which an attacker with a user credential can discover another user's bookmark list. There are 65 different docs. Each user can have up to 7 bookmarks. The worst case is choosing 7 out of 65 which requires many observations (65! /(7! * 58!). There is clear side-channel in space: the. packet sizes of the

bookmark pages differs for users. The dynamic analysis of other teams show that there is a large number of secret leakage. However, the leakage seems not be large enough to recover current bookmarks of a user as the buckets are too large (given the 200 operations). We did not observe a side-channel in time, which could have provided more leakage. Litmedia2 is definitely more vulnerable compared to Litmedia1 (question 56)."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.11 Phonemaster

A.7.1.1.2.11.1 Question 003 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there a side channel vulnerability in time from which an attacker can discover the username of any user (target user) who submits any request demonstrated in the script. At most, there can be 150 users. 100 users are fixed in the database. SimulateUsers can add at most 50 users to the database (usually it is about 20-30). Attacker, in offline mode, can experiment the existence users in the system and obtain their running time. After running the SimulatreUsers.sh file that includes 48requests, the attacker first sorts the running time for those that are in increasing order. These timings are related to new users added to the database and they are added in the order. The remaining running times are related to the users have already existed in the system (the 100 ones) or the repeated query for the new one. Since the attacker has profiled the existence users in the offilne mode, he can exactly discover those users. For the any new user, the attacker can also discover their ids based on the locations of the users in the system. We use our neural network tools and find out a strong leakage from the secret inputs to timing observations. Our neural network model obtains the public and secret inputs and learns the timing model of program (non-functional aspects). We use a neural network of 1,000 neurons with barbarized outputs. The results of the network found 51 classes of observations that are the indication of strong information leaks."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.12 Powerstate

A.7.1.1.2.12.1 Question 031 (AC Space, Intended Not Vulnerable, Answered: Yes)

Response:

"Yes, there exists an algorithmic complexity vulnerability in space. An exploit was claimedby the Northeastern team, but we could not confirm it. The CodeHawk analyzer showed several complexmethods (see Q037), which generate large matrices that are transformed etc. We did not have sufficienttime to explore this further."

Evaluation:

Confirmed NE exploit for an unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.12.2 Question 037 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there exists an algorithmic vulnerability in time. An exploit was claimed by the North-eastern team, but we could not confirm it. The vulnerability resides in the methodcom.bbn.powerstate.PowerStateSolver.solve.Another exploit was reported by the Grammatech team, who were not able to actually run the exploitdue to the inclusion of Java 9 classes (Module.java, etc.) which their platform did not support.This method was also identified by our CodeHawk analyzer as the most complex method. The loopsummary below shows two methods with high complexity:LoadFlowSolver.solvewith 22 loops and amaximum nesting level of 4, andPowerStateSolver.solvewith also a maximum nesting level of 4. TheLoadFlowSolver.solvemethod is not reachable, so we can ignore it. ThePowerStateSolver.solvemethod is called fromRequestHandler.handle. The downstream call graph from this method, shown inFigure 7, indicates that several of the other high complexity methods are called as well from the handlemethod, includingState.generatemeasurements,State.verity,PowerFileParser.parse, and theconstructor for thePowerStateSolverclass. To determine the complexity of thePowerStateSolver.solvemethod in terms of input sizes, we create a cost model for this method […] in nanoseconds. Many methods have been profiled and their cost is includeddirectly (if constant). The cost of other methods is shown as symbolic cost values with and indicationof the methods referred to. The lower-bound and upper-bound clearly show the dependencies on thevarious parameters of the method, and would perhaps allow an optimization to find the largest possiblevalue within the budget constraints. Due to the limited time available we did not perform such anoptimization.The costmodel for thePowerStateSolverconstructor is somewhat simpler, but with heavy dependenceon the number of lines

Evaluation:

Confirmed the exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.13 PSA

A.7.1.1.2.13.1 Question 010 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there exists a vulnerability in time. Iowa created an exploit that works as follows: first submit a POST request for an analysis that does not exist. That will create a lock file in that directory. Then submit 200 GET requests for that analysis. A thread will be allocated for each of these, but will block indefinitely because of the lock file, thus exhausting the thread pool of 200 threads, and thus the server is not able to service any more requests after this."

Evaluation:

Did not find the intended vulnerability; confirmed exploit for unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.13.2 Question 030 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"Yes, there exists an algorithmic complexity vulnerability in space. The Grammatech team demonstrated that when the server is presented with a json file that consists of 3200 square brackets followed by a 0 followed by 3200 closing square brackets will result in a 31MB file being printed. The vulnerability exists in the method PyDisassembler.start. Figure 8 shows the control flow graph for this method; writing the file to disk happens at pc=264. The control flow graph shows that no sanitization is performed before this point, so any valid json file can reach this point. A json file can be presented by the user to the server. This json file will be processed with only json format validation and is then pretty printed. The resulting indentation is quadratic in the number of square brackets, thus resulting in a large file."

Evaluation:

Not exactly the intended vulnerability; Confirmed this unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.14 Roulette 1

A.7.1.1.2.14.1 Question 013 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"No, there is not side channel vulnerability in space and time. The winning number depends on two components and the side-channel in space and time are too weak to reveal it to the attacker."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.14.2 Question 019 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"No, there is not an algorithmic complexity vulnerability in space. The server only writes to the file playersDB.ser in the datadirectory of the challenge problem. The existing file is roughly 2MB in size and consists of two parts: 1) data about the players (key, status) and 2) general statistics(connects, disconnects, number of bets, number of winners). It is possible to slightly increase the second part, but the budget of 10kb in way too small to get the playersDB.ser to 1Gb."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.14.3 Question 048 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"No, there is no side channel vulnerability in space. We did not observe a clear side channel. We also inspected the code and nothing stood out. We wanted to construct a proof, but lack of time prevented it."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.14.4 Question 054 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"No, there is algorithmic complexity vulnerability in time. The attacker would be able to block access to the server if somehow 10000 bets could be placed in the queue. During the collaboration we concluded that the budget was (way) too small."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.15 Roulette 2

A.7.1.1.2.15.1 Question 011 (SC Space/Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there is a side channel vulnerability in space and time. GrammaTech demonstrated now the winning number can be computed before the betting is off. The demonstration uses a side-channel in time that allows combining the prior winning number and the rotation (measurable in time in advance). We observed that the standard deviation for computing the angle is 0, but we did not have the full picture before GrammaTech showed it to us."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.15.2 Question 033 (SC Space, Intended Vulnerable, Answered: No)

Response:

"No, there is not a side channel vulnerability in space. We discovered a small side-channel in space using Wireshark. However it is too weak to discover another user's bet."

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.1.2.15.3 Question 046 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"No, there is not an algorithmic complexity vulnerability in space. The server only writes to the file playersDB.ser in the datadirectory of the challenge problem. The existing file is roughly2MB in size and consists of two parts: 1) data about the players (key, status) and 2) general statistics(connects, disconnects, number of bets, number of winners). It is possible to slightly increase the second part, but the budget of 10kb in way too small to get the playersDB.ser to 1Gb."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.16 Securgate

A.7.1.1.2.16.1 Question 008 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"Yes there is a side channel in space. The sum of all the packets sent from the server to the client as image blocks after requesting an plate provides enough information to determine which plate was requested. The transcribed text sent back is also determined by the plate.

This side channel was determined experimentally. For a given plate, the sum of packet lengths corresponding to image block transmission was one of 13 values for each plate. These possible values for each sum can be determined ahead of time. The attacker can measure this value to determine the plate sent. But, the value measure may correspond to one or more plates.

These packet sums were determined for a subset (about 1/8) of the possible plates. In this subset 3% of the values overlapped, resulting in more than one possibility of plate being sent. Extrapolating to the full set of plates, about 22% of values are overlapping. Also, it is extremely unlikely that any value will have more than 8 plates corresponding to it. Thus the attacker can use 1 passive operation to observe the session and determine the sum of packet lengths corresponding to the plate. This gives up to eight plates that the attacker thinks may have been sent. Then up to seven oracle queries can determine which plate was sent. The attacker also knows the transcribed text corresponding to each plate, and therefore knows the license plate number sent from the client back to the server."

Evaluation:

Correct; confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.16.2 Question 018 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Our initial proposed exploit was the following. By setting the transparency bit we create an image of 9000 by 9000 that is within the budget of 10kB. After a client receives a plate id, together with the size of the plate, the client submits requests for all the blocks that belong to that plate. A block is 8 by 8, hence there will be a million block requests for this single plate. All these requests will be queued up within the server. We assume the server has 200 threads, each of which can service a request. For each block request there is a 5 secs delay (Thread.sleep(5000)), resulting in most threads sleeping most of the time. A thread can sleep at most 500 secs. So we have to keep the server saturated with block request. A request from a different client (the benign request) has to compete with all of these block requests. Assuming 200 threads, we need to compute the probability that the benign client request is not selected for 100 cycles of 5 secs. After discussion with Grammatech and Iowa we decided that this exploit is out of scope, since we do not know that the attacker has the capability to add an image of his own to the video feed. We now believe the attacker does not have the possibility from within a client to delay the benign request by 500 secs."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.17 Suspicion 1

A.7.1.1.2.17.1 Question 020 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"No, there is no algorithmic complexity vulnerability in space that would cause the challenge program to store a file with a logical size 2GB given the resource usage of 2 kB. The only method that can cause this vulnerability is unzip method in MorseCode class. Attacker can store a large file with passing large positive and negative integers. However, recordQuantity method does not allow for passing a negative value. Without passing the negative values, the attacker is not able to store a file of size 2GB in the given resource usage limits."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.17.2 Question 026 (SC Space, Intended Vulnerable, Answered: No)

Response:

"No, there is no side channel vulnerability in space from which an attacker can determine the location before it is announced no matter how the other players answer any of the questions. Note that there are at most 25 different possible locations. The application encodes the location information in base 3 and represents them with three indices. These indices are stored in a array where each element can have values from 0 to 2. The total length of packets in each step of the game is determined by the one index. Lets say that the index array is E. Then the length of packet in the first step is a constant value plus E[0], the second step is a constant plus E[1], and in the third stage is a constant plus E[2]. Each unique value from 0 to 25 can be encode with unique array value E. The vulnerable packets are crafted in MorseCode class in the outcome array. The grabLength method inside DistrustCompetition class is the parts of application that use outcomearray to determine the length of packets. However, we did not observe the differences in the experiments. We always observe the same packet sizes of 1056bytes. We believe the application completely pad the size side channels.

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.1.2.17.3 Question 040 (SC Space/Time, Intended Vulnerable, Answered: No)

Response:

"No, there is no side channel vulnerability in space and time from which an attacker can discover the secret password. The length of password is at most 5, and there is no limitation on the characters that can be chosen. Codehawk helped us to figure out the conditions on the secret: 1) the length of password is less than equal to 5, 2) there is no restriction on the values of chars."

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.1.2.18 Suspicion 2

A.7.1.1.2.18.1 Question 022 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"No, there is no side channel vulnerability in space and time from which an attacker, assigned as a spy, can discover the secret password. Password matching is safe. There is possibility of RSA attacks that allow an attacker, by analyzing the timing of a decryption, to tell if a given number is close to a multiple of one of the RSA primes. The attack is due to an extra reduction step during the Montgomery multiplication used in the modular exponentiation. However, the attack requires more than 60,000 operations and is not in the budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.18.2 Question 044 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"Yes, there an algorithmic complexity vulnerability in space that would cause the challenge program to store a 2GB file given the input budget 2kB. The vulnerability is in the expand method. The attacker can control what the victim is going to receive. The victim writes the data to a file in the method. The condition in the expand method can be bypassed with crafting requests in certain ways. It can be done by sending the requests with sequence ('.','-2000000000','.','+2000000000'). The attacker needs to modify the code inside AskIssueCommand.java to send the request from the client side."

Evaluation:

Confirm exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.18.3 Question 057 (Null SC Space, Intended Not Vulnerable, Answered: Yes)

Response:

"Yes, there is a side channel vulnerability in space from which an attacker, assigned as a spy, can determine the location before it is announced no matter how the other players answer any of the questions. This vulnerability is similar to the one insuspicion1question 026. Note that there are at most 25 different possible locations. The application encodes the location information in base 3 and represents them with three indices. These indices are stored in a array where each element can have values from 0 to 2. The total length of packets in each step of the game is determined by the one index. Lets say that the index array is E. Then the length of packet in the first step is a constant value plus E[0], the second step is a constant plus E[1], and in the third stage is a constant plus E[2]. Each unique value from 0 to 25 can be encode with unique array value E.

The values of these three indices determine the length of packets in each step of game. Unlike thesuspicion1question 026, we observe different packet sizes in the experiments."

Evaluation:

Colorado incorrectly determined there was a strong enough side channel in this variant.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.1.2.18.4 Question 060 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"Yes, there is an algorithmic complexity vulnerability in space that would cause the challenge program's memory usage to exceed 2GB limit given 2000 bytes. The vulnerability is in the expand method. The attacker can control what the victim is going to receive. The victim writes the data to a file in the method. The condition in the expand method can be bypassed with crafting requests in certain ways. It can be done by sending the requests with sequence ('.','-2000000000','.','+2000000000').The attacker needs to modify the code inside AskIssueCommand.java to send the request from the client side."

Evaluation:

Confirmed exploit for unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.19 ThermoMaster

A.7.1.1.2.19.1 Question 021 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"No, there is no algorithmic complexity vulnerability in space. The only file to write into islog.txt. The overhead for sending one request from the client is307+126+110=543 bytes, leaving us only 957 bytes for the actual contents to stay under the input budget. Hence we can at most send 2 requests. Each request can increase the size of the log file by at most approximately 608KB, and 2×608KB is below the 5MB resource limit."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.19.2 Question 038 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"No, there is no side channel vulnerability in time. All but one loops in this program have a bound linear to the size of input. The loop that is not linearly bounded by input size is bounded by a constant of 32 million, which is very complex and we did not find any way to have this loop run a significant different amount of iterations when given two secret inputs."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.19.3 Question 052 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there is an algorithmic complexity vulnerability in time. The vulnerability is inside a loop in method com.ainfosec.thermomaster.PID.run. Although the loop bound is a constant, the value of this constant is large, i.e. 32 million. Therefore, performing expensive float computation in each loop iteration is time-consuming. The exploit works as follows. It first disables all constants. Then it sets the temperature to the value of -FLOATMINVALUE, followed by setting the temperature to the value of FLOATMINVALUE. CodeHawk reports a suspicious loop in method com.ainfosec.thermomaster.PID.run whose bound is a huge constant (i.e. 32 million),

Evaluation:

Confirmed exploit; different from intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.20 Wordshare

A.7.1.1.2.20.1 Question 002 (SC Space, Unintended Vulnerable, Answered: No)

Response:

"No, there is no side channel vulnerability in space from which an attacker with the user key can discover the plaintext of any secret word in a target document. We observe that there is non-interference separation between secret data and public data. However, we observe slight differences in the packet sizes in sending add command with using encode mode. Upon further observations, we figured out that the differences are related to client side applications."

Evaluation:

Was originally correct but an unintended vulnerability was found.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.1.2.20.2 Question 005 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

"No, there is no algorithmic complexity vulnerability in space that would cause the challenge program to store a file with a logical size of 35 MB under the input budget of 250 KB. We provide two reasons for this: The first observations is that the total size of log files in the framework can't be more than 10MB. The second observation is that the only way to exceed the resource usage under the allowed input budgets is by sending multiple output files each separated with ':'. This ends up having more than one output file. However, the question requires to have only one output file."

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.1.2.20.3 Question 047 (SC Time, Unintended Non Vulnerable, Answered: No)

Response:

"No, there is no timing side channel that can leak the plaintext of any secret word in any document. The first possibility is the leakage of the secret key of the server. An attacker can obtain the secret key and decrypt the secret document. However, we do not observe any vulnerability in the encryption parts of the application. The second possibility is to recover the secret text from the TrieNode. For this, we need to distinguish prefixes that match with the prefix of the secret from other prefixes. Even though, methods such as findValueSuggestions seem to contribute in timing leakages, we were not able to observe timing differences between secret-matching prefixes and all other prefixes."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.21 YapMaster

A.7.1.1.2.21.1 Question 015 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"Yes, there is an algorithmic complexity vulnerability in space. The vulnerability can easily be triggered using the client: Simply log in and get the (non-existing) yap with the highest possible ID (900000). This creates a map file of 115MB), which is larger than resource usage limit.

The vulnerability is due to the procedure MemoryMapReader.get, which start reading at location (ID−1)∗128 bytes in the map file of the alias without first checking whether a yap with the ID actually exist. This causes the map file of the alias to be extended to (ID−1)∗128. This vulnerability was detected by interaction with the client and observing the size of the files in the maps directory."

Evaluation:

Colorado identified the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.21.2 Question 017 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"No, there is not a side channel vulnerability in time. Quite some information about the most recent yap can be observed by analyzing the network traffic (we used Wireshark for this purpose). The network traffic reveals the user, alias, and ID of the most recent yap as well as the encrypted content. However, we did not observe a side-channel in time that would allow us to decrypt the content. We collaborated with all the teams on this question, but nobody was able to find a side-channel. However, GrammaTech observed something interesting. It is possible overwrite the most recent yap, but using "user:alias" as alias and posting ID times yaps. However, this is probably an unintended operation and is not related to a side-channel."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.1.2.21.3 Question 042 (SC Space, Intended Vulnerable, Answered: No)

Response:

"No, there is not a side channel vulnerability in space. Quite some information about the most recent yap can be observed by analyzing the network traffic (we used Wireshark for this purpose). The network traffic reveals the user, alias, and ID of the most recent yap as well as the encrypted content. However, we did not observe a side-channel in space that would allow us to decrypt the content. We collaborated with all the teams on this question, but nobody was able to find a side-channel. However, GrammaTech observed something interesting. It is possible overwrite the most recent yap, but using "user:alias" as alias and posting ID times yaps. However, this is probably an unintended operation and is not related to a side-channel."

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2 GrammaTech

A.7.1.2.1 Answer Statistics

Table A-145: Engagement 7 GrammaTech Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	11	7	64
SC Time	13	11	85
SC Space/ Time	7	4	57
AC in Space	18	17	94
AC in Time	11	11	100
Total	**60**	**50**	**83**

Table A-146: Engagement 7 GrammaTech Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	31	23	74
Not Vulnerable	29	27	93
Yes Answer	26	24	92
No Answer	34	26	76

A.7.1.2.2 GrammaTech Specific Responses

A.7.1.2.2.1 Calculator 3

A.7.1.2.2.1.1 Question 023 (AC Space, Unintended Vulnerable, Answered: Yes)

Response:

“We believe that this attack exploits the inefficiency of the subtraction code and the division code (which in turn calls the subtraction code). Both LargeNumeral.subtract and LargeNumeral.divide contain relatively long-running loops that maintain long arrays of coefficients; as their loops execute, the methods both incrementally append coefficients of a LargeNumeral to the arrays, which leads to relatively inefficient use of memory.

The attacker has complete control over the mathematical calculation that the calculator performs; this allows the attacker great freedom in exercising the vulnerable parts of the code. We developed an input expression with the goal of exercising the subtraction code and the division code of calculator_3. Our expression is of the following form:

"(10^499)/2-(10^499)/2+…+(10^499)/2-(10^499)/2"

in which the expression "(10^499)/2-(10^499)/2" is summed 250 times. We measured the memory usage of the server as it served this request (on a laptop, not on the NUC) and saw that it went above the threshold.”

Evaluation:

Confirmed exploit for unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.1.2 Question 025 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

“We believe that a vulnerability is not present. The application is a calculator that performs mathematical calculations as requested by the attacker. This question asks about the possibility of causing the calculator application to write a large file to disk. The method AbstractHttpGuide.handle writes exception tracebacks to a log file when an exception of type ExecutionException or InterruptedException occurs during a call to Future.get(). The particular Future object on which get is called is a wrapper that wraps most of the HTTP-request-handling code of the application, including the code that performs the mathematical calculations requested by the user. An uncaught exception that occurs during the execution of the Future will result in an ExecutionException. Thus, one way to cause data to be written to the log file is to send a mathematical expression to the calculator that causes an exception to be thrown and ultimately caught in AbstractHttpGuide.handle. Knowing this, we investigated the question of what exceptions can be thrown by the calculation-handling code, and we eventually focused on the Calculator.handleExpression method.

The Calculator.handleOutput method is potentially vulnerable because it performs a recursive call to handle any parenthesized sub-expression of the expression that it is analyzing. Thus, when attacker supplies an expression having deeply nested parentheses, handleOutput will perform deep recursion; at a sufficiently great depth, a stack overflow exception is thrown that results in an ExecutionException, which causes a traceback to be written to the log file. Because the recursion is deep, the traceback itself is very long.

We investigated the possibility of using this stack overflow exception to craft a full attack. We ran a few expressions with a large number of nested parentheses and found that the log file was being written to. However, the log file grew only by a few kilobytes and was therefore a far way away from exceeding the threshold of 3.5GB. Querying the server with enough requests to get the log file to exceed the threshold using this exploit would thus far surpass the budget of 60KB.

For this reason we conclude that an attack based on this exploit does not fit within the given budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.1.3 Question 034 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"We did not find a vulnerability during the take-home engagement, but after the live engagement, we do believe that there is a vulnerability. We believe that this attack exploits the inefficiency of the subtraction code and the division code. In particular, the relevant methods are LargeNumeral.subtract and the method LargeNumeral.divide, which calls LargeNumeral.straightDivide, which calls `LargeNumeral.subtract.

The code for LargeNumeral.subtract contains randomized while loops that slow down its execution. Let the two numbers to subtract be X and Y, and let N be the number of decimal digits in the smaller of X and Y. Then, the body of the inner loop will be executed at least N times. But, the expected number of times that we reach the head of some loop is over 4N. While the program is executing these nested loops, it will spend some time executing the inner loop body; however, it will also waste time generating random numbers and going through many conditional branches at each level of the loop nest. We believe that this wasted time is the source of the vulnerability. Note that the expected amount of time wasted in this loop nest is proportional to N. Furthermore, this code is called in a loop inside straightDivide, which is called in a loop inside divide; the loops in those methods further amplify the amount of time wasted in the randomized loops of subtract.

Colorado found an exploit that uses a mathematical expression containing many division operations applied to large numbers; the divisions then exercise the subtraction code. The expression uses the Roman numeral calculator because Roman numerals allow large numbers to be expressed concisely. The expression is generated by the following snippet of Python code:

```
expr = '(X^LD/' + '/'.join('(IX^D' for I in range(100)) + '/III^D)' + (')'*100)

expr  = '^'.join([expr ]*13)"
```

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.2 Calculator 4

A.7.1.2.2.2.1 Question 036 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

"We found a vulnerability during the take-home portion of the engagement. We believe that this attack exploits the inefficiency of the subtraction code and the division code (which in turn calls the subtraction code). Both BigNumber.subtract and Bignumber.divide contain relatively long-running loops that maintain long arrays of coefficients; as their loops execute, the methods both

incrementally append coefficients of a BigNumber to the arrays, which leads to relatively inefficient use of memory. The attacker has complete control over the mathematical calculation that the calculator performs; this allows the attacker great freedom in exercising the vulnerable parts of the code. We succeeded in using the same attack for calculator_4 that we had developed for calculator_3's memory-usage question; we developed this input expression with the goal of exercising the subtraction code and the division code of calculator_3. We created an expression of the following form:

"(10^499)/2-(10^499)/2+…+(10^499)/2-(10^499)/2"

in which the expression "(10^499)/2-(10^499)/2" is summed 250 times. We measured the memory usage of the server as it served this request (on a laptop, not on the NUC) and saw that it went above the threshold."

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.2.2 Question 043 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"We believe that a vulnerability is not present. Our investigations focused on the method BigNumber.subtract and the method BigNumber.divide, which calls BigNumber.straightDivide, which calls `BigNumber.subtract. We investigated these methods because our tool IST identified them as part of the highest-ranked "hotspot" that it found in the code. We attempted to exercise these parts of the code by manually testing the server's responses to a variety of expressions. In our experiments, we were unable to achieve long running times. We also browsed the decompiled code, but did not find a way to exploit the code. One reason that we were interested in these sections of the code was that the corresponding methods in calculator_3 contain a vulnerability. Notably, however, the randomized loops inside the subtract method of calculator_3-- which we believe are important for the vulnerability there-- are not present in the subtract method of calculator_4."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.2.3 Question 045 (AC Space, Intended Vulnerable, Answered: No)

Response:

"We believe that a vulnerability is not present. The application is a calculator that performs mathematical calculations as requested by the attacker. This question asks about the possibility of causing the calculator application to write a large file to disk. The method AbstractHttpGuide.handle writes exception tracebacks to a log file when an exception of type ExecutionException or InterruptedException occurs during a call to Future.get(). The particular Future object on which get is called is a wrapper that wraps most of the HTTP-request-handling code of the application, including the code that performs the mathematical calculations requested by the user. An uncaught exception that occurs during the execution of the Future will result in an ExecutionException. Thus, one way to cause data to be written to the log file is to send a

mathematical expression to the calculator that causes an exception to be thrown and ultimately caught in AbstractHttpGuide.handle. Knowing this, we investigated the question of what exceptions can be thrown by the calculation-handling code, and we eventually focused on the Arithmatizer.processClause method.

The Arithmatizer.processOutcome method is potentially vulnerable because it performs a recursive call to handle any parenthesized sub-expression of the expression that it is analyzing. Thus, when attacker supplies an expression having deeply nested parentheses, processOutcome will perform deep recursion; at a sufficiently great depth, a stack overflow exception is thrown that results in an ExecutionException, which causes a traceback to be written to the log file. Because the recursion is deep, the traceback itself is very long. We investigated the possibility of using this stack overflow exception to craft a full attack. We ran a few expressions with a large number of nested parentheses and found that the log file was being written to. However, the log file grew only by a few kilobytes and was therefore a far way away from exceeding the threshold of 2.5GB. Querying the server with enough requests to get the log file to exceed the threshold using this exploit would thus far surpass the budget of 60KB. For this reason we conclude that an attack based on this exploit does not fit within the given budget."

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2.2.3 CyberWallet 1

A.7.1.2.2.3.1 Question 006 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"There does not seem to be any way to increase the memory consumption of the application significantly. Our tools identified locations where allocations take place and ranked them using heuristics."

Evaluation:

Correct but minimal justification

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.3.2 Question 035 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"The tier of an account is determined by the balance. There is a vulnerability located in com.cyberpointllc.stac.cyberwallet.manager.ArchiveInfoManager' in the method 'handleFetch. This methods builds the response to a user requesting the information of a bank account. The method uses the balance of the account to determine the variable picker:

long balance = archive.obtainBalanceInUSCents() / 100000L;

double rand = (double)balance + Math.abs(bankerAssistant.pullEnsuingRandom());

int picker = (int)Math.log10(rand);

This computation has a positive random component but it is very small compared to the balance. For example, in the first tier, balance will be between 0 and 9 and between 10 and 99 for the second tier.

The random component has a standard deviation of 1, so it will be less than 1 (in absolute value in the 68% of cases). This is because pullEnsuingRandom() calls nextGaussian(). Given that, the variable picker with take the following values:

Bronze picker=0

Silver picker=1

Gold picker=2

Platinum picker>=3

If picker is greater than 3, it will become 3. So we have a fixed picker number for each tier.

Finally, in BankerAssistant.takeEnsuingAd the picker number is used to select an add image from the list of add images. In particular, picker determines the second index of the ad*.gif files.

The size of these files follows a particular pattern that allow us to distinguish them in the traffic.

|Files with the second index | Size |

| 0 | 1.9 mb|

| 1 | 2.1 mb|

| 2 | 305 kb|

| 3 | 1.5 mb |

These sizes completely dominate the amount of information transferred during a session where the benign user checks his or her account. Therefore, by observing the traffic of a benign user checking the account we can infer the tier of the account.

In the worst case, a user can have a balance that is very close to the limit. E.g. a balance of 9999 which will make

the variable balance (in the code above) take the value 9. Then, the random value could affect the final value of picker by increasing the value of balance in more than 1 (and thus the file selected) with a probability of 32% (68% falls within a standard deviation which is 1). So we could get the wrong tier by one with 32% of probability.

The probability that we get the wrong tier by two tiers, we would need a random value that is much higher than two which will happen with a probability much smaller than %5 (This is enough because the required probability of success is 95%)

Once we have made one observation, we know that the real tier is the one that we obtained with 68% probability or one lower with 32%. We can use the second available operation and perform an oracle query to make sure which of the tiers is the right one."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.3.3 Question 051 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“In order to transfer money from one account one has to know the account internal id. This internal id (of 23 alphanumeric characters) is what is used as a path in the transfer queries. If the attacker knows the id, he does not need to login with the credentials of the target user, the system apparently does not check that the logged in user own the account when making a transfer

However, there is another way. If the user has the proper authorization code, she can transfer a negative amount of money to another account (effectively transfering money to her account). In the class com.cyblerpointllc.stac.cyberwallet.mananger.ExchangeCore we see that in the constructor, the field authorized is initialized with -1 if BankerAssistant.isAuthorized returns true

this.authorized = BankerHelper.isAuthorized(Integer.valueOf(authorization), this) ? -1 : 1;

Then we have in the method grabQuantity that the amount to transfer is multiplied by the field authorized

```
public long grabQuantity() {
    return (long)this.authorized * this.quantity;
}
```

The method isAuthorized compares the given authorization number to a fixed authorization number that is established at the start of the application and is a number between 0 and 100. We can easily try all the possible authorization codes until we find the one that changes the sign of the amount.

Finally, in order to make a transfer with a negative amount, we need to know the destination account number. There is a potential side channel in time present in the class com.cyberpointllc.stac.cyberwallet.ArchiveOverseer in the method locateArchiveIdentity. The amount of time this method takes will vary greatly depending on how close the queried face is to an existing face (A face here is an account number).

In constrast to cyberwallet_2, the side channel is not visible from the user’s perspective. Before the method 'locateArchiveIdentityis called, there is an additional check that stops the process if the destination account does not exist incom.cyberpointllc.stac.cyberwallet.manager.VerificationManager`, in the method makeTransfer. So we cannot try to find a destination account incrementally. Note that if we know the destination account that we want to take money from, we still can do it but there is no side channel involved.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.4 CyberWallet 2

A.7.1.2.2.4.1 Question 024 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

“The balance is used to determine the ads images that are displayed to the user and the sizes of the ads images vary significantly. The other related values cannot affect the amount of the network communication significantly.

Statements on why the application does not contain a side channel (strength and reachability).

In constrast to cyberwallet_1, the index used to select the ads image is computed as follows:

long worth = account.pullWorthInUSCents() / 100000L;

double rand = (double)worth * Math.abs(bankerHelper.getEnsuingRandom());

int seed = (int)(Math.log10(rand) % 6.0D) + 4;

The balance (worth variable) is multiplied by a random number which is going to make the selected ad vary randomly as well. Therefore, the side channel is not strong enough to infer the tier of the account given the limited budget of 2 operations.

Moreover the seed is also incremented by 4 so the numbers will be between 4 and 10. The images in that range have a much larger variability in size.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.4.2 Question 059 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

“In order to transfer money from one account one has to know the account internal id. This internal id (of 23 alphanumeric characters) is what is used as a path in the transfer queries. If the attacker knows the id, he does not need to login with the credentials of the target user, the system apparently does not check that the logged in user own the account when making a transfer. The account id only seems to be visible when you login with the owner user and there seems to be no side channel to obtain it.

However, there is another way. If the user has the proper authorization code, she can transfer a negative amount of money to another account (effectively transfering money to her account). In the class com.cyblerpointllc.stac.bankit.mananger.TransactionValue we see that in the constructor, the field authorized is initialized with -1 if BankerHelper.isAuthorized returns true

this.authorized = BankerHelper.isAuthorized(Integer.valueOf(authorization), this) ? -1 : 1;

Then we have in the method takeQuantity that the amount to transfer is multiplied by the field authorized

```
public long takeQuantity() {
    return (long)this.authorized * this.quantity;
}
```

The method isAuthorized compares the given authorization number to a fixed authorization number that is established at the start of the application and is a number between 0 and 100. We can easily try all the possible authorization codes until we find the one that changes the sign of the amount.

Finally, in order to make a transfer with a negative amount, we need to know the destination account number. To obtain that we use a side channel in time present in the class com.cyberpointllc.stac.cyberwallet.AccountConductor in the method findAccountFace. The amount of time this method takes will vary greatly depending on how close the queried face is to an existing face (A face here is an account number).

The side channel is sufficiently strong because the account number is checked character by character so it allows for an incremental search (from left to right). For each character, the method testForDuplicateError iterates over all the accountFaces (account ids) which takes a significant amount of time making the time difference noticeable."

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.5 Door Master

A.7.1.2.2.5.1 Question 029 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"In class keyfobcontroller we have a suspicious piece of code:

```
   @RequestMapping(method={RequestMethod.POST}, value={"/keyFobs/auth/"},
consumes={"application/x-www-form-urlencoded"})
   public String authKeyFob(@RequestParam(value="encKey") String encKey) throws
GeneralSecurityException {
      if (KeyFobService.KeyExists(encKey).booleanValue()) {
         System.out.println("Key exists, authenticating: " + encKey);
         String result = KeyFobService.Auth(encKey);
         while (result == null) {
            result = KeyFobService.Auth(encKey);
         }
         return result;
      }
      return "" + encKey + " does not exist.";
   }
```

The idea is to make this loop run forever by requesting /keyFobs/auth and deleting the key simultaneously. This can make one thread run forever. This is because KeyForService.Auth will return null if the id is not in the keyidMap. We successfully managed to make one thread loop forever, but the application is multi-threaded. We would need to exhaust the thread pool which does not seem feasible withing the 2kb budget. Therefore, our final answer is NO."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.5.2 Question 041 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"The sleep between pings in the method run of class MessageLoop contains a side channel vulnerability in time. If this.flag is true, the interval between two pings will be 100 or 200 miliseconds depending on each of the bits of the key of the last fob added to the application.

```
public void run() {
    do {
        try {
            KeyFobService.conn = this.ping();
            this.flag = KeyFobService.flag;
        }
        catch (InterruptedException interruptedException) {
            // empty catch block
        }
        try {
            Thread.sleep(this.flag != false ? KeyFobService.nextInt() : 1000);
        }
        catch (InterruptedException interruptedException) {
        }
    } while (true);
}
```

This is because the class bbn.Crypto.SecureRandom stores a string with the seed in binary in the field temp. The method engineGenerateSeed takes one of those bits each time it is called. The seed is updated everytime a fob is created with the private key of the fob.

The key is generated with RSA with a modulus of 512 which yields a key that is usualy around 343-346 bytes (2744 bits). The number of packets we can observe is much higher. We can observe the key twice

to make sure of its actual length and then use that number of active operations to try all the possible rotations of the key (in case we are unsure of where the key starts and where it ends).

Learning the private key is enough to gain access to the target user's id."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.6 EMU 6502

A.7.1.2.2.6.1 Question 004 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"There could be a side channel if the attacker was able to guess the admin password, which is required to access the AdminView page that would enable him to delete a user. To log in, the application is using some standard methods from the Vaadin and Spring frameworks.

There does not seem to be a way in the application code to learn the admin password. There does not seem to be a way in the application code to learn the admin password."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.6.2 Question 012 (SC Space, Unintended Vulnerable, Answered: Yes)

Response:

"When running the simulator, the client queries the server every ~1s to update the current value of the registers. The server sends a message with the updated values, which generates network traffic that is observable. Because the victim cannot change the initial values for the simulated program, there is only a single possible run for a given .s program. This run is deterministic, and every time the program will be simulated, the registers will have the same values across runs at each given time point. We have monitored the network traffic when running the same .s program twice, and filtered only the relevant packets.

We can see that the sequence of packet sizes is very similar after we ignore some initial packets that are likely due to timing differences (in the above table, there is an initial packet of size 1831 in run 2 that does not have a match in run 1).

The size of the packets depend on the values of the registers at the given time in the simulation. For this reason, the sequence of packet sizes are different if we simulate another program.

As a consequence, the attacker can run all the avaliable .s programs offline to generate those packet size sequences, and monitor the network traffic of the victim to identify which program he is running."

Evaluation:

Justification correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.6.3 Question 014 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"The method run of CodeExecutionThread in EmulatorView.java is used to emulate the program given as input. It contains a loop that is bounded by the size of the input program (EmulatorView.this.codeTable.getItemIds()). The loop body has a call to Thread.sleep(1000), which makes it simple to monitor the number of loop iterations, as the Vaadin application will send network packets every ~1s until the loop terminates.

Because the number of loop iterations depends on which program has been selected by the user, and because the user is not able to change the initial input values (they are reset to their default value before starting the program, even if the user tries to change them beforehand), we believe that this gives a sufficient signal for the attacker to guess which program has been run.

To get a stronger signal, it is also possible to monitor the time between packets: when the simulator is emulating a branch instruction, the time difference between two packets is larger. Again, because the inputs of the program cannot be changed, the attacker can pre-compute at what time the assembly code will execute a branching instruction."

Evaluation:

Justification correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.6.4 Question 016 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Because this is a Vaadin/Spring application, the server is using a pool of threads where each different user will have its own thread. The application could be vulnerable if we found a way to exhaust the number of threads or the memory of the resource, or if there was a synchronized method that takes a long time. There is no synchronized method in the application code. The attacker knows only a single user/password pair, which prevents him to exceed the number of threads in the pool. Moreover, the code is written is such a way that the user can run a single emulation at a time.

Based on these observations, we believe that there is no ACV-T in this application."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.7 InAndOut 1

A.7.1.2.2.7.1 Question 007 (AC Time, Unintended Vulnerable, Answered: Yes)

Response:

"There exists an algorithmic complexity vulnerability in time that allows an attacker to delay a benign request beyond the given budget. The relevant methods that contain the vulnerability are InAndOutAid.pickupOrder and ProcessedOrders.getBinaryCodeFromValidationCode, which is called in the body of InAndOutAid.pickupOrder. These methods are called when a user picks up a pizza. These methods are also relevant because the majority of the body of InAndOutAid.pickupOrder is wrapped in a synchronized block. Thus, if the attacker can busy up an execution within this synchronized block then the rest of the application will block any further requests. […]

The steps of the attacks are as follows:

1. First an attacker orders a pizza order. The actual order does not matter. The application will respond with random validation code. For example say the returned code is 1-0-4-8-0-32.
2. The attacker then modifies the given validation code to a different string. However, the modified validation code must correspond to the same binary code as the original validation code. This can be achieved by modifying any non-zero digit to a different non-zero digit. For example, using the previous example code, the attacker will choose a new validation code of 1-0-5-8-0-32.
3. The attacker will then pickup a pizza with the chosen validation code. For example, the attacker will pickup a pizza using code 1-0-5-8-0-32. The application will convert this value to a binary code via ProcessedOrders.getBinaryCodeFromValidationCode, which will be the same code as the one corresponding to 1-0-4-8-0-32. Thus, InAndOutAid.pickupOrder will not return because the binary code is in use. However, when the application tries to pick up this order an infinite loop will occur because the validation code 1-0-5-8-0-32 is not in use."

Evaluation:

Confirmed exploit for unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.7.2 Question 027 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

"There exists an algorithmic complexity vulnerability in space that allows an attacker to cause the memory usage of the application to go beyond the given budget. This algorithmic complexity vulnerability comes from a very specific pizza order. […]

Essentially, if an attacker makes an order of

"2,3,5,7,11" 7

that is 7 pizza's with tomato, pepperoni, onion, extra-cheese, and pineapple, then the application will call the constructor of OrderDirector, which will spawn a new thread which will eventually call CookingCONTROLLER.calculateStageSeries which will loop for a very long time on this particular input. Each iteration of this loop will add to an in use data-structure, which will thus cause the memory usage of the server to grow well past 2GB. Note that the application will respond back to the attacker quite quickly. This is because the thread that services the request will finish quickly; however, the thread that calculates the stage series will run for a very long time."

Evaluation:

Confirmed this exploit is the intended

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.7.3 Question 049 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"There exists a side channel vulnerability in time, which allows an attacker to pick up a benign user's pizza. The secret for this question is the validation code of the benign user. An attacker can obtain this information by observing the time it takes the server to generate a validation code

for a given binary code. The attacker can observe this computation time by observing the traffic between the serverNUC (Server) and the clientNUC (InventoryManager).

When a benign user orders a pizza InAndOutAid.situateOrder calls OrderTaker.takeOrder, which calls OrderTaker.notifiyStockCooking. This method sends a packet to the inventory manager, which is the first observable event the attacker notes. After this notification OrderTaker.takeOrder calculates a validation code from a binary code using OrderTaker.calculateValidationCode. Shortly after the calculation of the validation code, InAndOutAid.situateOrder calls OrderTaker.scheduleOrderStock, which in turn again calls OrderTaker.notifiyStockCooking. This method again sends a packet to the inventory manager, which is the second observable the attacker makes note of. The observable is the elapsed time between these two message. Note that the only variable time code that executes between these two messages is OrderTaker.calculateValidationCode. In other words, the execution time of OrderTaker.calculateValidationCode is the observable which reveals the secret. […]

The side channel is sufficiently strong because out of the computations that happen between the two calls of OrderTaker.notifiyStockCooking, OrderTaker.calculateValidationCode is the one with variable time. This is exactly the observable we need to find the secret. From the Vanderbilt team we also have some experimental evidence that shows the tight relationship between the elapsed time between the packets sent form the server to the inventory manager and validation codes. They showed that consecutive validation codes have a difference of 50ms timings with very little overlap. Thus, we conclude that an attacker would be able to distinguish validation codes via this observable.

Originally we noticed the potential side-channel leak through the computation of a validation code from a binary code via our viewing of the decompiled source code. However, we were unsure if this leak was sufficiently strong for the attacker. During the collaborative part of the engagement the Vanderbilt team shared some experimental results that showed the elapsed time between the two messages sent from the server to the inventory manager was directly proportional to the validation code generated for that order. They showed that consecutive validation codes have a difference of 50ms timings, which is enough time difference to extract the validation code. During the collaborative part of the engagement we were able to determine that the code that server executes between the two packets sent from the server to the inventory manager was the generation of a validation code from a binary code. Thus, their experimental results confirmed our original suspicions that the execution time of generating a validation code leaks the generated validation code."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.7.4 Question 055 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"We do not believe there is a side channel in space which allows an attacker to pickup a benign user's pizza. The attacker can observe the size of packets sent between the server and the benign user as well as the size of packets between the server and the inventory manager. The packets between the server and the inventory manager contain values representing the pizza toppings the

user ordered as well as some other random values. Thus, we do not believe the secret (validation code) relates to the size of these packets.

As for the packets between the server, these packets contain the secret (validation code). The packets sent between the server and the user does contain the generated validation code, which can vary between 11 and 13 characters. Thus, it may be possible to learn some information about the length of the benign user's validation code. There does not seem to be any way to get enough information about validation codes from packet sizes. The packets to the inventory manager do not reveal any information, and the packets to the user do not reveal enough information to determine the validation code within the budget. We arrived at this conclusion through manual analysis of the decompiled source code. Our ideas were strengthened when the Vanderbilt team presented some additional empirical evidence that showed packet sizes don't obviously correlated with secret values."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.8 InAndOut 2

A.7.1.2.2.8.1 Question 028 (SC Time/Space, Intended Vulnerable, Answered: No)

Response:

"We do not believe there to be a side channel vulnerability in space and/or time that allows an attacker to pick up a pizza of a benign user. […] As far as space vulnerabilities, this application uses the same packet structure as inandout_1.

The attacker can observe the size of packets sent between the server and the benign user as well as the size of packets between the server and the inventory manager. The packets between the server and the inventory manager contain values representing the pizza toppings the user ordered as well as some other random values. Thus, we do not believe the secret (validation code) relates to the size of these packets. As for the packets between the server, these packets contain the secret (validation code), but the packet size does not reveal enough about the secret to yield an attack.

As far as time, we developed an attack for inandout_1 based on the elapsed time between the two packets sent from the server to the inventory manager. This attack was based on the execution time to convert a binary string to a validation code.

We do not believe there exists any side channels based on space information which leak the validation code within the application.

It is possible that inandout_2 reveals some information about the validation code in a similar way as inandout_1. That is an attack based on the amount of time it takes to convert a binary code to a validation code. We state in the next section why we do not believe this side channel to be strong enough to reveal the secret.

We also don't believe there are any other timing side channels within this application. Because there aren't any other narrow enough observable events that an attacker can measure to determine the amount of execution time for a couple of methods. In other words, the elapsed

time between other observable events is filled with too many variable computations which would therefore weaken any side channel leaks.

For the take home portion of the engagement we answered yes, believing the exploit for inandout_1 would work for inandout_2. However, at the collaborative engagement Vanderbilt presented the referenced empirical evidence which allowed us to conclude that no side channel exists for this application."

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2.2.8.2 Question 039 (AC Time, Intended Vulnerable, Answered: Yes)
Response:

"There exists an algorithmic complexity vulnerability in time that allows an attacker to delay a benign request beyond the given budget. The relevant methods that contain the vulnerability are InAndOutAid.pickupOrder and ProcessedOrders.takeBinaryCodeFromValidationCode, which is called in the body of InAndOutAid.pickupOrder. These methods are called when a user picks up a pizza. These methods are also relevant because the majority of the body of InAndOutAid.pickupOrder is wrapped in a synchronized block. Thus, if the attacker can busy up an execution within this synchronized block then the rest of the application will block any further requests. […]

The steps of the attacks are as follows:

1. First an attacker orders a pizza order. The actual order does not matter. The application will respond with random validation code. For example say the returned code is 1-0-4-8-0-32.
2. The attacker then modifies the given validation code to a different string. However, the modified validation code must correspond to the same binary code as the original validation code. This can be achieved by modifying any non-zero digit to a different non-zero digit. For example, using the previous example code, the attacker will choose a new validation code of 1-0-5-8-0-32.
3. The attacker will then pickup a pizza with the chosen validation code. For example, the attacker will pickup a pizza using code 1-0-5-8-0-32. The application will convert this value to a binary code via ProcessedOrders.takeBinaryCodeFromValidationCode, which will be the same code as the one corresponding to 1-0-4-8-0-32. Thus, InAndOutAid.pickupOrder will not return because the binary code is in use. However, when the application tries to pick up this order an infinite loop will occur because the validation code 1-0-5-8-0-32 is not in use.

Evaluation:

Confirmed Exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.8.3 Question 053 (AC Space, Intended Not Vulnerable, Answered: No)
Response:

“We do not believe an algorithmic complexity vulnerability exists which allows an attacker to cause the server to use enough memory to exceed the budget. A potential vulnerable method for this application is CommercialPROCESSOR.computeStateSeries. The reason this method is potentially vulnerable is because it has a loop that might run a variable amount of times and for each execution and integer would be added to a data structure that is in use. Since the data structure would be live within this method the memory used would not be free, and the memory usage of the application would monotonically increase. Dynamic evaluation of code surrounding CommercialPROCESSOR.computeStateSeries revealed that the loop within this method always executes a constant number of times for any pizza order. In other words, for any possible pizza order the amount of memory used by CommercialPROCESSOR.computeStateSeries is constant.

Because the potentially vulnerable method only uses a constant amount of memory for any possible pizza order, we conclude that this application does not contain an algorithmic complexity vulnerability.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.9 Litmedia 1

A.7.1.2.2.9.1 Question 009 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)
Response:

“We believe there is an algorithmic complexity in space in the IconGuide class. When a user requests an HTML page, the HTML page often contains a link to an image. The client has to send a HTTP GET request to obtain that image. When the server receives such request, it executes the above code.

The above code is reading the image from the disk and puts in in a BufferedImage object. Then, it writes this image into a ByteArrayOutputStream stream. Then, this stream is converted into a Byte array.

All these operations are expensive in terms of memory usage.

The attacker can send multiple HTTP GET requests to download images from the server, in a short time interval. This will create many parallel executions of the above code, each of them allocating memory for the images.

In total, this will exceed the budget of 600Mb.

We tried the attack and monitored the network traffic to make sure it does not exceed the input budget.”

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.9.2 Question 032 (SC Time, Intended Not Vulnerable, Answered: No)
Response:

“We have identified that some computation triggered by a call to grabPublishingFile, which is reachable in the callgraph from PublishingViewerGuide.handleGrab (i.e. when the victim opens a new piece of literature), may be expensive.

This grabPublishingFile reorders the list of picks by calling the method updatePicks, which has some recursive calls. These recursive method calls are computing a bubble sort. There could be a sidle channel if this bubblesort computation depends on the secret data.

The secret information is actually not the entire PickNode objects, but only the fields base and tags in these objects. However, manual inspection of the code, (or, taint analysis if we want some automation) can show that these fields are actually not used in the bubble sort, except for computing the hashcodes for PickNodes. By looking at the hashcode implementation of PickNode it is clear that this cannot lead to any significant leakage.

We could combine several approaches to get a higher confidence in our ‘No’ answer:

We can use Vanderbilt’s network profiling that computes statistics on the packets timings to show that there is no significant differences that could leak the secret information

We can define the secrets as taint sources in IST, and look at the taint analysis results to show that only the results of the hashcodes are tainted in the bubble sort computation.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.9.3 Question 056 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

“The method of interest is handleGrab in the class BookmarkGuide. This is the method that is used to build the response to the victim’s request to view its bookmarks page. When looking at this code, we can see that the size of the page depends on the content of the stringBuilder sb. For each bookmark, a new line is inserted in the resulting HTML page.

As a consequence, we believe that this program leaks some information about the victim’s bookmarks because the size of the returned HTML page varies depending on the bookmarks. For instance, experiments showed that the size of the HTML page is 95 bytes larger after adding “A Christmas Carol” to the bookmarks list. This 95 bytes is exactly the size of the text mentioned above.

More generally, for each bookmark, the size of the page will increase by 77 + the length of the book’s title. The length of the book’s titles vary between 4 and 41. This means that the leak could almost allow to discover how many books are saved as bookmark. However, this leak does not seem to be sufficient to leak this number precisely, nor it is able to determine exactly which books are bookmarked.

Another source of potential leak is also the execution time of the call to takePublishingFile. This method is called one for each bookmark, and is used to get the book’s information from its identifier. The books are stored in a hashmap implemented in the MapDB library.

We can see that the method serialize has been randomized, but the method deserialize is not randomized. From this, we deduce that the execution time of the call to takePublishingFile might

leak some information about the number of tags assigned to the book. Still, we believe this signal is not strong enough to implement an attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.10 Litmedia 2

A.7.1.2.2.10.1 Question 001 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"Similarly to litmedia_1 Question 032,

The secret is the list of picks private List<FavoriteNode> favorites; in the class MediaLibraryImpl, which is the ordered list of top-picks of the victim user. When the victim is requesting a new piece of literature, this list of picks is updated. There could be an side channel if the values of the FavoriteNode objects inside the favorites can be learned from looking at the network traffic between the victim and the server. Our tools have not identified any potential code fragments where we could identify a vulnerability."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.10.2 Question 050 (SC Time, Intended Not Vulnerable, Answered: Yes) Are saying Yes here but wrote "N" on their answer form

Response:

"The observable is the time taken to process the request to view a piece of literature. The secret is the id of the piece of literature that was requested by the victim. We believe that there is a side channel in time that allows the attacker to guess the number of lines in the requested piece of literature.

The relevant line is the line that iterates over the lines in the file that contains the content of the piece of literature:

br.lines().forEach(line -> sb.append((String)line).append("
"));

There is another loop for (int p = 0; p < takeLabels.length; ++p) { that can also make the execution time vary. As a consequence, the total execution time will depend on both the content of the piece of literature, but also on the number of tags assigned to this piece.

The probability of success has to be 90%, and the input budget is 80, which allows 7 oracle queries, so we believe that this side channel is strong enough to answer 'Yes' to this question.

Evaluation:

They answered Yes based on the justification

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2.2.10.3 Question 058 (SC Space/Time, Intended Not Vulnerable, Answered: Yes)

Response:

"The relevant entry point is the method handleGrab in MarkerBroker. This method is called when a user is accessing its bookmarks page. This method returns a webpage, whose size is going to have some dependencies on the secret, which is the list of markers (i.e. bookmarks) returned by the method call to library.fetchMarkers().

For each marker, some elements are atted to the templateMap and the stringbuilder sb. These objects are then used to build the final HTML page. The size of this HTML page is observable by looking at the size of the network packets.

As a consequence, we believe that this program leaks some information about the victim's bookmarks because the size of the returned HTML page varies depending on the bookmarks. For instance, experiments showed that the size of the HTML page is 95 bytes larger after adding "A Christmas Carol" to the bookmarks list. This 95 bytes is exactly the size of the text mentioned above. More generally, for each bookmark, the size of the page will increase by 77 + the length of the book's title. The length of the book's titles vary between 4 and 41. This means that the leak could almost allow to discover how many books are saved as bookmark. However, this leak does not seem to be sufficient to leak this number precisely, nor it is able to determine exactly which books are bookmarked. For this reason, we initially answered 'No' to this question. However, during the collaboration, some teams have actually tried all possible combinations of bookmarks, given that there is a maximum of 7 bookmarks and 65 documents. For each possible combination, the attacker can compute offline what is going to be the size of the resulting HTML page. From there, we learn that some combinations of bookmarks result in the same HTML size as other combinations, but given that there is a budget of 200 operations, one can use up to 199 oracle queries to disambiguate between the various possibilities, and one only need a accuracy of 80%. It turns of that in most cases, only 60 oracle queries are sufficient to identify the secret information.

The attacker can compute the size of every possible bookmark page offline. Then, the attacker can passively observe the network traffic, and then ask oracle queries to disambiguate between possibilities."

Evaluation:

They claim a SC-S, with length of bookmarked media titles (apparent in the packet sizes) revealing which media files are bookmarked. This is not possible. Colorado gave a solid combinatorial argument for the existence of a worst-case secret for which the oracle queries would be sufficient, and the control team identified one.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2.2.11 Phonemaster

A.7.1.2.2.11.1 Question 003 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"The secret is the user making the request. We can check the list of users in order with the get userlist interaction. When a user makes a request, the method CheckLogin in the class Server is called. The checkPassword method is very expensive and it is computed for all the users in the list until the right user is found.

The time spent checking the login is proportional to the position of the user in the list i.e. there is a side channel in time. We can measure the response time of a request and infer the user that made the request from it.

CheckLogin has a random waiting time PAD_MILL but this time is always the same so we can observe several interactions to infer it. We can add a new user and check how much longer it takes to login for that user. That way we can know the time needed for executing checkPassword once. Then we can infer how many times checkPassword was called in the target request."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.12 Powerstate

A.7.1.2.2.12.1 Question 031 (AC Space, Intended Not Vulnerable, Answered: Yes)

Response:

"Most of the computation in the program takes place in the class PowerStateSolver, in particular the method solve which takes an instance of State and tries to find a solution.

This method creates and operates over several Matrices based on the input data. We believe that some of the matrices can become very big based on the number of buses.

We can provide an input file with many buses that is still within the budget. One of the partners during the live engagement provided a vulnerable input file that triggers this response."

Evaluation:

Confirmed that exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.12.2 Question 037 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Some network descriptions do not have a solution an it can take a long time to make the analysis. Running this example takes above 1 minute.

This example is a modified version of the original example where the phase angles of the branches have been restricted so the problem is not solvable. n principle it would seem that that is not enough to make the code vulnerable because the web server used is non-blocking. However, the key is as the beginning of the method PowerStateSolver.solve. This code counts the instances that are currently running and it compares it to the number of available processors. Therefore, if we can create more instances that processors are available, the following requests will be kept waiting at this point.

The file above has a size of 2519 bytes, which means that we can send approximately 20 requests within the budget of 50KB. This is well above the number of cores in the target platform."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.13 PSA

A.7.1.2.2.13.1 Question 010 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"During the take-home engagement, we did not find a vulnerability, but during the live engagement, we did find a vulnerability during a cooperative discussion with several teams.

During the take-home engagement, we identified several suspicious parts of the code, but we did not find any that could be exploited within the constraints of question 010. In particular, an attacker may be able to cause PSA's deserialization module to deserialize attacker-controlled input files; if so, that is also a serious vulnerability. We noticed that the attacker can use the "modules_to_load" parameter to cause the deserialization module to run. The attacker can use another parameter to change the directory from which files are deserialized. The attacker should be able to use this parameter to cause the module to deserialize files from an attacker-controlled directory, such as the "/apps" directory, to which the attacker can upload files using an HTTP PUT request. The attacker could use this upload facility upload various well-known "deserialization bombs" which cause slow execution and other undesirable behavior. In particular, we were interested in the possibility of a deserialization bomb that would exhaust memory or other resources on the server and thereby cause a denial of service.

There are probably several ways to exploit PSA's static analysis routines directly, by submitting carefully crafted Python programs. PSA's analysis code contains various static analysis routines that have high computational complexity. Some analysis parameters, such as the number of values to store in a value analysis, or the length of contexts to use in a context-sensitive analysis, appear to be attacker-controllable, although we did not figure out how to change them. Ultimately, PSA's analyzer code was not easy to exploit within the small network usage budget of this question. A particular problem was that even when a small Python program was crafted, the corresponding decompiled Python file (in JSON format) has a size much greater than the network usage budget. Moreover, the attacker would still face the problem of how to slow down many threads at once, without consuming too much of the network usage budget. For this reason, we concentrated most of our investigations on less resource-intensive approaches to slowing down threads of the server.

In the intended workflow, analysis of a Python file creates a "LOCK.ME" file which gets deleted when the analysis completes. However, an attacker can run an analysis that will create its "LOCK.ME" file and then crash, leaving the "LOCK.ME" file in place indefinitely. The attacker can do this in several ways, for example by sending an HTTP PUT analysis request for a Python file that does not exist on the server. This creates the "LOCK.ME" file the project directory for that analysis. Because the requested Python file doesn't exist, the analysis will then crash without deleting its lock file. Then, the attacker can send subsequent HTTP GET requests to the same project directory and these GET requests will block indefinitely, waiting for that lock file to be deleted.

If the attacker sends 300 such GET requests, they eventually exhaust the netty thread pool with threads that are blocked indefinitely. Finally, after that thread pool is exhausted, the benign request to the web server's root will also be blocked indefinitely. Thus, we answer "yes" to question 010.

Experimentation by Iowa, in collaboration with all other teams, showed that this attack could be performed with approximately 70KB of network traffic, which is well below the 80KB budget."

Evaluation:

Not the intended vuln; confirmed exploit for unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.13.2 Question 030 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"The vulnerability is present in the start method of the PyDisassembler class. This code receives a JSON file from the client and writes a pretty-printed version of the same file back to disk using the Gson library.

Pretty-printing of a JSON file adds spacing characters to produce a cleanly indented file. If the nesting depth of the data structures in a JSON file is unlimited, then the indentation added by pretty-printing amounts to a quadratic increase in file size in the worst case.

The PyDisassembler class does not check the validity of the JSON file that it is given before it pretty-prints that file to disk. As a result, the attacker can send a completely invalid file that is designed to cause a large size increase after pretty-printing. In particular, the attacker can send a file consisting of 3200 pairs of square brackets around a zero. This file is less than 7KB in length (which is less than the network traffic budget of 8KB), but the pretty-printed version of it is larger than the 20MB threshold specified in the question."

Evaluation:

Not the intended vuln; confirmed exploit for unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.14 Roulette 1

A.7.1.2.2.14.1 Question 013 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"The winning number is computed in the class RouletteWheel in the method spin. This computation depends on:

1. The time the roulette spins (timeRotate)
2. The last position of the roulette (lastRotateAngularSeparation)
3. Several parameters of the roulette that are fixed in the class MatchParameters (such as the frenchWheelAngularSpeed and the wheelTimeOffset)

The second and third dependencies we can observe directly from the code and the previous winning number. The first one, the timeRotate is computed in the method Game.getTime. This variable is random and is not related to any observable. Therefore, there seems to be no side channels that allow predicting the result beforehand"

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.14.2 Question 019 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"The only part of the code that seems to write to disk seems to be the class ParticipantsDB which contains a map of participants Id and Participant. This map can be grown by adding new participants to the database but the increase in file size is linear.

A suspicious element is the implementation of the class ParticipantsList which allows for two users with the same id to connect simultaneously. This leads to strange behavior in how the bankroll and the bets are managed but I could not find a way to take advantage of such behavior to increase the database size.

In order to increase the size to 1Gb with a budget of 10KB it would be necessary to add many participants. But there seems to be no other way of adding participants than connecting to the server with a new ID. In this manner it is not possible to reach 1GB with only 10KB of budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.14.3 Question 048 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"The secret are the type, value and amount of the bet performed by another user.

This information is transferred through UPD (sic) packets between the user and the server. The information is contained in the request and also in the response and the size of the packets (the observable) is affected by the content.

The code that deals with a bet request is in the class PlayersExchange, in the method handleReceivedMessage.

The only communication that takes place is the one for placing the bet, receiving the response and the broadcast about the winners. The size of the packets of placing the bet is related to their content but this does not seem to provide enough information to infer the complete content.

It does not seem possible for the attacker to perform any actions that will have a response from the server containing information about other people's bets. There amount of information that has to be infer is quite significant, there are many possible bets, compared to the information that one can extract from the network traffic."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.14.4 Question 054 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"This request is handled in handleReceivedMessage in the class PlayersExchange which is called in the class Secretary in the private class MessageHandler that has its own thread. It does not seem to be any way to block or delay this request.

There is some suspicious code. The blocking queue for the wagers can block for a long time if we reach the maximum number of bets. This happens in stac.roulette.BetList.

The capacity is 10000 and the queue is created once. so even if the Quantities.capacity is modified later (it is modified in DozenWager), this does not affect the capacity of the queue.

We cannot place 10000 bets with 10KB of budget, so we cannot trigger this response.

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.15 Roulette 2

A.7.1.2.2.15.1 Question 011 (SC Space/Time, Intended Vulnerable, Answered: Yes)

Response:

“The winning number is computed in the class RouletteWheel in the method spin. This computation depends on:

1. The time the roulette spins (timeSpin)
2. The last position of the roulette (lastSpinAngularGap)
3. Several parameters of the roulette that are fixed in the class GameParameters (such as the frenchWheelAngularSpeed and the wheelTimeDisplacement)

The second and third dependencies we can observe directly from the code and the previous winning number. The first one, the timeSpin is computed in the method Game.takeTime which adds the variable lastTime and a random number.

The field lastTime is assigned in the method Game.wagersOn and is equal to the variable wagerReminderTime.

This variable is in turn used to schedule a reminder message "Remember to place your bet!".

Therefore, the delay between the message “Bets are on!” and “Remember to place your bet!” in seconds corresponds to the value of lastTime which constitutes a side channel in time.”

Evaluation:

They found the intended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.15.2 Question 033 (SC Space, Intended Vulnerable, Answered: No)

Response:

“The secret are the type, value and amount of the bet performed by another user.

This information is transferred through UPD packets between the user and the server. The information is contained in the request and also in the response and the size of the packets (the observable) is affected by the content.

The code that deals with a bet request is in the class PlayersExchange, in the method handleReceivedMessage. The only communication that takes place is the one for placing the bet, receiving the response and the broadcast about the winners. The size of the packets of placing the

bet is related to their content but this does not seem to provide enough information to infer the complete content.

It does not seem possible for the attacker to perform any actions that will have a response from the server containing information about other people's bets. We performed some network profiling to check the sizes of the packages and examined the decompiled code to make sure the information about the bets does not leak anywhere else."

Evaluation:

Incorrect; they missed the intended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2.2.15.3 Question 046 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"The only part of the code that seems to write to disk seems to be the class PlayersDB which contains a map of players Id and Player. This map can be grown by adding new players to the database but the increase in file size is linear.

A suspicious element is the implementation of the class PlayersCatalog which allows for two users with the same id to connect simultaneously. This leads to strange behavior in how the bankroll and the bets are managed but I could not find a way to take advantage of such behavior to increase the database size.

In order to increase the size to 1Gb with a budget of 10KB it would be necessary to add many participants. But there seems to be no other way of adding participants than connecting to the server with a new ID. In this manner it is not possible to reach 1GB with only 10KB of budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.16 Securgate

A.7.1.2.2.16.1 Question 008 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"We believe there is a side channel leak in space which allows an attacker to determine the transcribed license plate text sent from the client to the server. […]

When the server has sent all of the blocks for an image the client will "transcribe" the image to determine the string written on the license plate. The client does this by running an Md5 hash on imageData.blockLenghts, which consists only of the sequence lengths from the response_data variables. The client determines the transcribed value via a lookup table on the Md5 hash. Once the client determines the transcribed value the client sends this value along with "id" and "token" to the server encrypted.

What 3 shows is that the transcribed value (secret) can be determined via the lengths of the JPEG blocks sent by the server to the client.

Traditionally one would not be able to determine the lengths of the JPEG blocks sent from the server to the client because these messages are encrypted and the encryption would vary the message sizes and obscure the plain text lengths. However, the server and client application use the following specification ``AES/GCM/NoPadding''. According to this link https://stackoverflow.com/questions/31851612/java-aes-gcm-nopadding-what-is-cipher-getiv-giving-me the ciphertext messages will be 12 + plainText.length + 16. In other words the encryption is not hiding the response_data lengths. This means an attacker can simply observe the message sizes for the JPEG blocks sent between the server and the client to determine the block lengths, which is enough to determine the secret."

Evaluation:

Unpadded the encrypted paddings for a correct mapping

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.16.2 Question 018 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"We believe that there is no vulnerability. This question asks whether a benign "/get_next_plate" query can be slowed down. We see no obvious vulnerable structures in the code that responds to this query. Another possible avenue of attack would be to exploit some sharing of state between the threads of the server; however, we did not find a way to do this.

The Colorado team developed a hypothetical exploit that works by creating a large image that has 9000 by 9000 blocks. Such an image can be fit into the network usage budget by making the image entirely transparent. The web framework seems to have a default thread pool containing 200 threads. Because the image is very large, it saturates the server's ability to send image blocks for the full 500 seconds. However, we believe that this exploit is out of scope because the attacker does not have a way to send this image to the server."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.17 Suspicion 1

A.7.1.2.2.17.1 Question 020 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"This question is similar to the question 044 of suspicion_2. However, the attack that works for suspicion_2 does not work here, because of an additional conditional that prevents from having a negative integer value in the encoding.

Indeed, if one of the num value is negative, the conditional if (sumQuantity < currentSum) { in recordQuantity will evaluate to true and an exception will be raised."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.17.2 Question 026 (SC Space, Intended Vulnerable, Answered: No)

Response:

"The location pointer (ie. the secret) is passed as argument to the method handlePointer in the method DistrustCompetition.generateAndSendOBjectives. This method call is going to initialize the array encoding in DistrustCompetition. As a consequence, the array encoding is tainted from the secret. In order to know exactly what is leaked, we have to look at the method encodePointer.

base is equal to 3, so s is going to be the base-3 encoding of the integer pointer (i.e. the secret), which means the string s has size 1, 2 or 3. As a conclusion, the array this.encoding will contain it its first 3 indexes the three digits of the base-3 encoding of the secret. Finally, encoding is used to compute the length of the message in grabLength. This length is then observable by monitoring the size of the packets. In the first three rounds, the code integer will leak the three digits of the base-3 encoding of the secret. The return value length of grabLength is finally used to send a message of size length.

As a conclusion, monitoring the size of the packets in the first three rounds of the game should allow the attacker to learn the value of the secret.

However, we have done some experiments while running the game and monitoring the network traffic. It seemed like when the game master is sending messages with the sendAll method, it generates one UDP packet per player and these packets all have the same size of 1024 bytes. This means that the side channel that we see in the source code is actually not observable. For this reason, even though our analysis of the code tells us there is a side channel, we decided to answer NO to this question."

Evaluation:

Saw the trie-vuln in the source code but determined it to be unreachable. Conclude this incorrectly

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2.2.17.3 Question 040 (SC Space/Time, Intended Vulnerable, Answered: No)

Response:

"We believe that the observables (time) does not relate to the secret. We have discovered that the method isPasswordCorrect has a potential for side channel. We can use a taint analysis to figure out which parts of the code are using the secret information, i.e. :

the value of final String password = objectiveMsg.getPassword(); in the GAME_ASSIGNMENT case in DistrustSecretary.handleReceivedMessage: this is the password received by all the players (except the spy, who receives a fake password)

the value of String password = joiner.toString(); in StartCompetitionCommand.execute: this is the string that receives the password just after is it typed by the game master.

We see from the taint analysis that the password is actually never really used in any way, except for the method isPasswordCorrect in DistrustCompetition. In particular, it is not stored by the players when they receive it (it is just printed).

We analyzed the method isPasswordCorrect to see if its exection time would depend on the secret password. If that was the case, the attacker could try to perform an attack by successively sending wrong passwords and monitor the execution time of the server until it returns an error. However, we see that the loop in isPasswordCorrect always iterates len times, when len is the length of the smaller password (either the secret password or the attempted one). At best, this would leak the size of the password, which is not strong enough to craft a working attack."

Evaluation:

Incorrect; missed the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2.2.18 Suspicion 2

A.7.1.2.2.18.1 Question 022 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"The observables are the timings of the packets and their size. Thus, the relevant methods generating observables are the calls to Suspicion.sendMessage. A possible way for the attacker to discover the secret password would be to measure the execution time of the method that checks whether the password he provides matches the correct password. The code is in DoubtTournament.java.

Because this code has a for loop that iterates len times, it seems possible for the attacker to guess the value of len, and thus the password size, with repeated messages with wrong passwords of different sizes.

Moreover, if the attempted password has the right size, the attacker could potentially learn how many characters are correct: the "then" branch of the if-then-else if (provided.charAt(i) != this.password.charAt(i)), which has 1 instruction match = false, is more expensive than the "else" branch, which has no instruction. However, because this instruction is very simple, we believe it won't be sufficient to get a side channel."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.18.2 Question 044 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"We answered yes to this question for the same reason as Q060. See answer to question 060."

Evaluation:

Correct due to vulnerability in 60.

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.18.3 Question 057 (Null SC Space, Intended Not Vulnerable, Answered: No)

Response:

"There is a difference in the code between suspicion_1 and suspicion_2, that make the side channel vulnerability inexploitable in certain situations.

We would expect that the value of this.plays is 0 for the first turn, 1 for the next turn, etc. However, it appears that the this.plays field gets incremented every time the method getPlays get called. In a typical game, the side channel attack would be feasible, if we assume that the benign users only ask questions in the first three rounds.

Depending on the kind of message received by the game master:

1. if the message is a question or an answer, the getPlays method is called only once.
2. however, if the message is for instance an accusation, getPlays will be called two times. Then, the server will send back a message to the sender, telling him that an accusation in the first three round is not permitted. The benign user now has to send another message. If it sends an accusation again, the server will again increment its counter plays. This means that, by then, this.plays will be above 2, and the value returned by grabMagnitude is not tainted by the secret anymore.

As a conclusion, the same side channel as Q026 exists here, if we assume that the benign users will only act "normally" in the game. However, we do not think that the side channel is strong enough in all generality, as it depends on the benign user's interactions.

Moreover, as also explained in the justification of Q026, it seems like the Netty library adds some padding to the final message in such a way that the packets send by the game master have a fixed size. This makes the side channel not observable, even though it appears to be present when analyzing the source code."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.18.4 Question 060 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"During the game, the players can ask questions to other players and answer those questions. The code is charge of sending the message that contains a question is implemented in the class AskIssueCommand. In this class, the raw message is first encoded using the MorseCode class.

final byte[] issueBytes = MorseCode.encode(issue);

Later, when the other player receives the message, its content is decoded using the instruction final String issue = MorseCode.decode(issueBytes); There is a vulnerability in the method expand: with some specific values returned by the input data stream, it is possible to create a very large file as well as a very large memory usage: the result of the decoded message is first stored in a temporary file tmpFile, and then the content of this file is read into a byte array byte[] encodedBytes.

The encoded message consist in a sequence of pairs (character, number of occurences), that tells what's the next character in the decoded message and how many times it occurs contiguously. For instance, the encoded sequence 'h',1,'e',1,'l',2,'o',1 is decoded into the message hello.

So, if the encoded message is 'c',10000, the decoded message will be of length 10000.

In the method expand, there is a conditional if ((aggregateContent += num) > 1000000) { that ensures that the message is no longer than 1000000. However, because num could potentially be

negative, there is a way to create a encoded message that will not trigger the exception "Data exceeded maximum allowed size", if we use a negative value for the first number. An example of such input is 'c', -2000000000, 'c', 2000000000. This input will generate a file of 2Gb and then a memory usage of 2Gb.

The attacker can use a modified version of the suspicion_2.jar file where the method execute of the AskIssueCommand class has been replaced. When the attacker will send a send_question command to another user, it will cause the ACV-S in both disk usage and memory on the victim's machine. This attack is within budget, as the network traffic induced by sending the above message is 1120 bytes."

Evaluation:

Confirmed this unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.19 ThermoMaster

A.7.1.2.2.19.1 Question 021 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"We do not believe there is an algorithmic complexity vulnerability in space that can be triggered by an attacker.

The logger gets text added at each iteration of the main simulate loop in PID.run. The loop iterates at most 32,000,000 times. However, values only get added to variables that will be written to the log file at every 1,000 iteration. In other words every 1,000 iteration of the main simulate loop contributes 1 line to the log file. Therefore, if the simulate loop executes at the maximum of 32,000,000 times then the log file will have 32,000 lines. It seems that there is no way to get the logger to write more than 32,000 lines per request.

As stated in the previous section the amount of logging data written to a file is bounded by the hardcoded loop bound in PID.run. We also note that the benign request causes the PID.run loop to execute almost all of the 32,000,000 iterations. In other words, we do not see how the attacker can cause a file to be written that is much larger than the one caused by the given example benign request.

Our sub-problem heuristics identified the loop inside PID.run as a potential vulnerability. Manual analysis of this loop allowed us to conclude that there is no way to get the loop to execute for more than 32,000 iterations, which is not enough to exceed the threshold given in the question."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.19.2 Question 038 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"We do not believe a side channel in time exists which allows an attacker to learn the set point value of a benign user. The only observable for this problem is the elapsed time between the

message sent from the benign user to the server and the response from the server back to the client. This elapsed time corresponds to the execution time to simulate the user's PID request. The execution time is depends on the value of the set point and current temperature sent from the user to the server.

As stated in the previous section the elapsed time between the benign user's request and response can reveal some information about the secret.

Other than this we do not believe there are any other side channels in the application. This is because the server creates new PIDs for each user. Thus, after the benign user makes their set point request this information will be stored in that user's associated PID. When the attacker makes a request a new PID with new values will be generated and associated with that attacker. Each PID simulation is run in a separate thread, thus it does not appear that the attacker can perform any active operations that will be influenced by the benign user's PID values.

For the remaining passive operation. We do not believe it is possible (in the worst case) for the attacker to determine the benign user's set point value. To see why consider a user who enter's a set point value very near their current temperature. The simulation loop will only execute a small number of times, because the error will become insignificant fairly quickly. The attacker will then learn this information by reading the elapsed time between request and response packets. However, the attacker is only learning that the set point is close to the current temperature. The actual set point however is indistinguishable from all the other set points. Thus, this passive operation is not strong enough for a full attack.

We reached this conclusion after a collaborative meeting with the other teams. We all determined that there are no useful active operations, and the passive operation is not strong enough for a full attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.19.3 Question 052 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"We show that there exists an algorithmic complexity vulnerability that allows an attacker to delay a benign request beyond the threshold.

The application contains very few loops. The main loop that takes a noticeable amount of time is the PID.run method. In this loop a thread is performing a sequence of PID simulations. This loop has a hard-coded bound of 32,000,000 iterations. Furthermore the benign example script runs for nearly all of these 32,000,000 iterations. Thus, at first glance it does not seem possible for an attacker to get this method to take more time then the example script. However, the calculations that are taking place inside the PID.run method are floating point calculations, which do not run in constant time for all inputs. Therefore, if an attacker is able to pick some floating point values for which computation is expensive they will be able to slow down the main simulation loop.

The code is also vulnerable in the sense that while the server creates new threads to run each simulation, the server thread has a Thread.join statement for it's simulation thread. This means the server thread will block while its child thread is simulating. Thus the server thread will be unable to service any benign requests during a simulation.

As stated previously, if an attacker can providing floating point values for which computation is expensive they could delay a benign request. Floating point values which yield expensive computations are ones that are small. The application prevents some small values as inputs with the PID.isValid method; however it does allow values which are floating point normalized. These values are quite small and yield quite expensive computations.

The following request message demonstrates the exploit:

request = “”

request = request + “C\n”

request = request + “C’spasswordpass\n”

request = request + “C\n”

request = request + “reset\n”

request = request + “enable windup\n”

request = request + “set setpoint\n”

request = request + "-1.175494351e-38\n"

request = request + “set current\n”

request = request + “1.175494351e-38\n”

request = request + “disable proportional\n”

request = request + “disable derivative\n”

request = request + “disable integral\n”

The set point and current temperature values are small floating point values which cause expensive computations, and disabling the proportional, derivative, and integral coefficients ensure the simulation will run for the whole 32,000,000 iterations.

We have experimentally found that the above request delays a benign request by approximately 20s on the reference platform.”

Evaluation:

Not quite the intended vulnerability; confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.20 Wordshare

A.7.1.2.2.20.1 Question 002 (SC Space, Unintended Vulnerable, Answered: No)

Response:

“A privileged user will add the target document containing at least one secret word.

When doing so, the attacker can monitor the size of the packet corresponding to uploading a document. However, because the secret word is already encoded when the document is sent, has the encrypted words all have the same size, we believe that this observable does not relate to the secret.

The attacker can also perform some active operations:

add a new document to the server: in that case, the server is going to return the string “ok” (unless there is an exception), which does not leak any information. If there is an exception, the exception message does not seem to leak secret information.

search for a word in an uploaded document: in that case, the server is going to return the array of documents that contain the searched prefix. This does not leak the secret either, as the secrets are filtered out by the server before sending the list of results.

Based on the above observations, we believe that there is no side channel in space in this application.”

Evaluation:

Was originally correct; another team found an unintended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.2.2.20.2 Question 005 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

“We can identify places in the code that write to a file. One way to write to files is through the logging mechanism. However, we believe that it is not possible to write a log file that exceeds 35Mb, due to the fact that the application is using sl4j, and the sizes of the log files using this library is limited to around 10Mb.

Another way to create files in this application is by uploading a document to the server. The document is then stored on disk on the server side. However, the content of the file is sent by the attacker through the network, and the budget for the attack is limited and well below 35Mb.

We also noticed that the method equals in the class WordShareDoc is also creating files on disk.

Manual experiments on the program showed that the above code can lead to a vulnerability when the attacker is doing the following:

add.sh file.html 0:1:2:3:.....999

add.sh file.html 1:2:3:4:5:....1000

This will create 1000 different files on the server side, all of them containing the content of the file file.html.

By uploading a small file file.html, and assigning a large number of aliases in the way it is done in the attack above, the server will write a large number of different files, which in total exceed 35Mb.

Given the formulation of the question, we understood that the server must contain a single file of size > 35Mb for the attack to be valid. This means the above attack is not in scope. Thus, we believe the answer to this question is no.”

Evaluation:

Correct according to their understanding of the question

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.20.3 Question 047 (SC Time, Unintended Non-Vulnerable, Answered: No)

Response:

“The attacker has no passive operations. The only two things he’s able to do is:

upload a new document and monitor the time before receiving the server’s response.

perform a search for a prefix and monitor the time before receiving the server’s response.

There could be a side channel in time if one of these two times depends on whether a given secret word — or the prefix of — is already stored in the T9 trie data structure that is used to store the existing words.

The relevant data structure in this application is T9Trie<WordShareDoc> trie in the class WordShareDocControl. More particularly, using his active operations, the attacker can try to monitor the execution time of:

trie.insertValue() if he adds a new documents.

trie.getT9ValueSuggestions() if he performs a search.

These two operations will have a different execution time depending of the current content of the Trie. We made some experiments to see if the time to perform a search on the server side would vary depending on whether a given prefix is in the trie data structure or not, but we could not find a situation where the time difference would be observable.”

Evaluation:

Correct.

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.21 YapMaster

A.7.1.2.2.21.1 Question 015 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

“The application keeps a record of all the messages that are sent in file. The classes that take care of that are

MemoryMapWriter and MemoryMapReader. For each user, the application creates a file and stores the messages according to their id. For that it maps each id to a region of memory starting at address (id-1)*128 and that extends 128 bytes (the maximum size of the messages).

In principle, it seems that all we have to do to increase the size of the file is to write a message with a very high id. However, messages are assigned id by the server incrementally and limited to 10000 (which is not enough) to achieve 100MB. The message modifications are stored with ids starting from 10000 and ending with 20000. For these ones we can start doing modification with ids 20000 and keep incrementing the id with new modifications one by one. However, we cannot achieve 100MB within budget like that. MemoryMapWriter is not vulnerable.

On the other hand, MemoryMapReader is vulnerable. We can try to read a message with id 900000 which will map a buffer to the (900000-1)*128 address. This will resulting a file with a size of about 115MB even if no writing is actually performed in the file.”

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.21.2 Question 017 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"There is a part of the code that is very suspicious. Mainly [how] YapUser.equals is implemented. That means that we can take the following alias friends:joey from the user brady and we will be sharing the cache with joey from the user friends. This is because the caches are stored in a list that uses equal to check if an element is in the list.

The relevant method is getCacheInstanceFromMaster in YapCache.

However, we do not share the YapRegistry so there seems to be no way of just reading the content of the cache. This is because YapRegistry are indexed in a hashmap that does not use the equal method but the object identity.

A possible attack is to write some messages with the alias friends:joey. That will write in the cache. Then we let the target user joey post some messages, these will overwrite the content of the cache. Finally we just have to read our own messages and we will see the content of the messages posted by the target.

The problem with this is that we need to post our messages before the target does which does not seem possible given the question description. Moreover, there is not side channel involved in this vulnerability."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.2.2.21.3 Question 042 (SC Space, Intended Vulnerable, Answered: No)

Response:

"Our answer for question 42 is the same as for question 17.

[From 17]:

The problem with this is that we need to post our messages before the target does which does not seem possible given the question description."

Evaluation:

Incorrect; the example script was provided to setup the minimum set of users and messages (secrets to be leaked) necessary for this application and did not stipulate when the setup occurs or if it was the only interactions that would occur on the server.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3 Iowa State

A.7.1.3.1 Answer Statistics

Table A-147: Engagement 7 ISU Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	11	6	55
SC Time	13	12	92
SC Space/ Time	7	5	71
AC in Space	18	17	94
AC in Time	11	11	100
Total	**60**	**51**	**85**

Table A-148: Engagement 7 ISU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	31	23	74
Not Vulnerable	29	28	97
Yes Answer	24	23	96
No Answer	36	28	78

A.7.1.3.2 Iowa State Specific Responses

A.7.1.3.2.1 Calculator 3

A.7.1.3.2.1.1 Question 023 (AC Space, Unintended Vulnerable, Answered: Yes)

Response:

"During take home session, we were able to locate several potentially vulnerable code and trigger 2.6 G memory usage with dynamic analysis. However, time was limited and we were not able to come up with an working exploit. Based on the findings and dynamic analysis results we have, we believed that there is some AC space vulnerabilities, but we doubt if we can make it under budget. So we answered no to this question. In live collaboration, GrammarTech shared their exploit with query like this: S=10^499 /2, and add up (S-S) for 250 times. They also claimed that this attack works on question 036 as well. Colorado shared their exploit to this question in this form:
99^518*99^518*99^518*99^518*99^518*99^500*99^500*99^500*99^500*999^478.If we send 72 such query to the server at the same time, it will trigger a memory usage more than 4G. GrammarTech used divided by 2 in their exploits, so they tried to reason that there is something wrong with the divide function. However, there was no direct evidence in the code that supports this idea. It also could not explain why Colorado's exploit works. According to our static analysis in the original report, we believed that it has something to do with the massive number of recursive calls in Reformulate.java, and the exploits support this hypothesis (they all applied huge amount of multiply operations). Other teams seems to have no objections to this hypothesis.

As we have two different working exploits from other teams and our static analysis showed some corresponding vulnerable code in the app, we decided to change our answer to YES”

Evaluation:

Confirmed exploit and unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.2 Question 025 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

“Theoretically, the log file size is limited to 10MB which means it will never get to 3.535G. So it is not possible.In practice, when we tried to trigger a log file writing to the server, we used large numbers of parentheses to perform a stack overflow error which writes the log file. We found that no matter how long the log information is, the log file only writes the first 1024 lines of logs which only add 94 kb to the file each time. To reach 3.535G file size, it takes 37,606 times of log writing and obviously will make the attack out of budget. Therefore, our answer is NO.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.2.1 Question 034 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

“During the take home engagement, we were able to locate several suspicious code segments in the source code, and trigger a long halt to the server with dynamic analysis. However, we were not able to successfully delay the benign query for more than 120 seconds. Therefore, we made a conclusion that the answer was NO. In the live engagement, GrammaTech team shared some promising results which may delay the benign query for up to 75 seconds. The query calculates a huge number then divides it by 2 for multiple times (10^499/2/2/.../2). However, none of the teams was able to get more than 120 seconds out of this. At the revisit session, Colorado team shared their possible exploit with all teams. The exploits takes advantage of Roman Numeral calculation and generates a huge number that the server cannot handle. People from our team verified that the exploit worked on NUCs. Unfortunately, our team noticed that even though it worked on NUCs, the exploit from Colorado is still slightly out of budget. After taking some effort, our team modified the query to make sure it is within budget and successfully delay the benign query for 121 seconds in the meantime. […] Together with the reasonable explanation to this vulnerability we had in the take home session, we decided to change our answer to YES with confidence.”

Evaluation:

Confirmed their exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.3 Calculator 4

A.7.1.3.2.3.1 Question 036 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

“After live collaboration on question 023, GrammaTech shared their exploit with query like this: S=10^499 /2, and add up (S-S) for 250 times. They claimed that this attack works on question 036 as well. Northeastern said that their exploit for question 043 works on this question. Colorado shared their exploit to this question: 999^351*999^351*999^351*999^351*999*351*999^333*999^333*999^333*999^333*999^246*99. If we send 70 of this query to the server at the same time, it will trigger a memory usage more than 4G. GrammaTech and Northeastern both used divided by 2 in their exploits, so they tried to reason that there is something wrong with the divide function. However, this could not explain why Colorado’s exploit works. According to our static analysis in the original report, we believed that it has something to do with the massive number of recursive calls in Multiply.java, and all three exploits support this hypothesis (they all applied hume amount of multiply operations). Till the end, no team has a better explanation to the vulnerability. As we have three differentworking exploits and our static analysis showed some vulnerable code in the app, we decided to change our answer to YES.”

Evaluation:

Iowa’s exploit can sometimes exceed the resource bound in the reference environment, the probability of success is approximately 36%, rather than the required 99% (based on 25 runs). Credit is given assuming they worked collaboratively with the other teams but may have not had time to update their answer.

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.3.2 Question 043 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

“In the live engagement, Northeastern team shared their exploit with the query “3^1049*115*2/2*2/2*...*2/2” (12,709 times of “2/2” in it). Their hypothesis is that the first two parts of the query generates a huge number. Then the last part of the query keeps multiplying and dividing the number by two which triggers the code which causes the vulnerability. However, after verifying the exploit, it turned out to be out of budget. The actual tcpdump for the query is 50 kb and the budget is 10 kb, so it does not work. During the collaboration, we had a working exploit for corresponding question 034. So we tried to apply the same strategy on this one. However, the same query does not do the same thing there which reason everyone was interested in. With some dynamic analysis using VisualVM, we found that most of the time was spent at divide() and subtract(). Given that subtract() is also called by divide(), we narrowed it down to subtract(). By comparing the same method between two apps, we found that calculator_3 have a highly nested while loops with random probability of access. The same code was replaced with a normal for loop here. Our hypothesis is that the randomness added to the code significantly increases the number of iterations that method runs through. By doing dynamic analysis, we found that it iterates 3.4 more times than normal for loop, and the Math.random() function call takes noticeable runtime during the process.As a result, all teams came to an agreement that it should be a NO answer to this question.”

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.3.3 Question 045 (AC Space, Intended Vulnerable, Answered: No)

Response:

"Theoretically, the log file size is limited to 10MB which means it will never get to 2.535G. So it is not possible.In practice, when we tried to trigger a log file writing to the server, we used large numbers of parentheses to perform a stack overflow error which writes the log file. We found that no matter how long the log information is, the log file only writes the first 1024 lines of logs which only add 94 kb to the file each time. To reach 2.535G file size, it takes 26,968 times of log writing and obviously will make the attack out of budget. Therefore, our answer is NO."

Evaluation:

Incorrect; they did not find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3.2.4 CyberWallet 1

A.7.1.3.2.4.1 Question 006 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"No evidence has been found for this question. And no dynamic analysis results supports that there exists such AC space memory vulnerability."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.4.2 Question 035 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"The secret here is the tier of an account -which is based on the account balance. The additional background says that the target user will sign in and check the account balance. This happens when the target user clicks on one of his accounts. The html page shows the account information. Our analysis points to the "ArchiveInfoManager" class which returns this webpage. The secret is either the "balance" (account balance) which can reveal the tier or the Archive tier itself which is printed on the page. Since this page contains the secret, it is possible that the length of the packets returned(observables) corresponding to this webpage somehow reveal the secret (account tier).

Taint on the "balance" variable reveals that it flows into BankAssistant.getEnsuingAd(picker), which helps to select a image to be displayed on the webpage. Our hypothesis is that which image is displayed can be used to determine the range of "amount" and hence the account tier. So the "picker" that is passed to BankAssistant.getEnsuingAd(picker), determines the image shown. The images are available to the analyst for offline analysis. So there is a potential side channel where the image displayed can reveal the balance and hence the tier. The images are available for offline analysis so by observing the network packets, it might be possible to figure out which image is displayed and after that the account balance. Our analysis shows that given a seed, with 95% probability we can narrow down the choices to two account tiers. We can then use one oracle query to figure out the correct account tier. Given one passive operation to observe the network traffic, we believe that we can get the secret in 2 observations. […]

…if the attacker can measure the ads picture in the communication, he will be able to get the class of the picture. Based on the class of the picture, with help of oracle queries, he will be able to get the correct account tier with success rate of 100%. Thus, we believe that there exists a side channel vulnerability in space with which the attacker can know the account tier of a target user."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.4.3 Question 051 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"No critical evidence has been found. 1.In VerificationManager.java, handlePost() method, there are two nested while loops may cause time delay for the post being handled. 2.handleDelivery() method of ConfirmationManager.java contains three nested while loops which may cause huge delay to the system. 3.However, no evidence has been found that those code is related to any side channel vulnerabilities. No evidence has been found to support a side channel attack. So we believe the answer is NO."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.5 CyberWallet 2

A.7.1.3.2.5.1 Question 024 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"During the live engagement, our team successfully located the vulnerable code with static analysis. Then, we were able to collect useful data with dynamic analysis. After applied some data analysis techniques, we realized that we can actually build a close relationship between the account tier and file name of promotion ads picture. In another question (question 035), we

did the same trick and the file size is clearly related to the file name and we can easily distinguish them accurately. However, in this question, the file sizes are literally randomized and has nothing to do with file name. We made a mistake in believing that file names are part of the side channel in space. So we believed that if we can figure out the account tier by file name with the help of oracle queries, the side channel exists. In live engagement, after discussed with other teams, although there was more randomness added to the seed generation part of code, we can still prove that we can reveal the account tier by file name with success rate greater than 95%. Other teams have no objection on our strategy. However, we still cannot build a strong relationship between file size and file name which means our strategy will have a success rate lower than 95%. Therefore, we decided to change the answer to NO after the live engagement."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.5.2 Question 059 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"In the take home session, we overlooked some important code segments which may relate to the vulnerability. During the live engagement, GrammarTech shared their working exploit with all teams. Their strategy consists of 3 steps: 1) blindly guess each digit of target account from bit_0 to bit_9, there is a response time difference when the attacker gets a correct guess; 2) make a transaction with some amount, using a randomly guessed authorization number from 0 to 99.3) if the confirmation page returns with a negative number, confirm it and finish the transaction. Otherwise, reject and repeat step 2) until we get a negative value.The code related to the vulnerability is shown below: 1)The testForDuplication() will always return false because testValue will never contain value. Also, testForDuplication() takes time because it will always go through the whole

loop. findAccountFace() checks the account number digit by digit. If first three digits are correct, it will stop at 4th. Thiswill make a response time difference in the communication which forms a side channel in time. This makes guessing account number possible for the first step. The account number is 10 digits long, 10 possibilities per digit, so the worst case is 100 operations to get a correct account number

Evaluation:

Confirmed GrammaTech's exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.6 Door Master

A.7.1.3.2.6.1 Question 029 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"At the end of the take home engagement we were exploring the possibility of disrupting pings between the server and client somehow, working under optimistic assumptions about what was possible. This approach was finally discarded during the live

engagement for various reasons, including the assumption that disrupting a ping response (assuming that it's even possible) from the position of the master NUC would probably have a lower probability of success than 50%.Northeastern noted that the KeyFobService keyidMap and myMap data structures are unsynchronized, and since is multi-threaded, there is a race condition between authKeyFob and deleteKeyFob.

The following sequence leads to an infinite loop:

1)Create a key

2)Start authorizing a key

3)Delete the key

If deleteKeyFob deletes the key after authorization runs KeyExists but before Auth, Auth will return null and the loop will run forever.Colorado put the above sequence in a script, and repeated the sequence until a server thread started running infinitely, proving this is possible. However, doing this even once will use up much of the input budget and doing it enough time to block all of the server threads is out of budget."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.6.2 Question 041 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"In take home engagement, we successfully located the related source code that leads to a vulnerability. However, we was not able to make a working exploit. Therefore, we marked this question no. During the live engagement, Colorado and Northeastern strengthened our confidence in our old hypothesis. Colorado was able to prove that the KeyFobService.nextInt() will only produce 100 or 200 as return value. With this hint, we know that the server will sleep between pings for 100 or 200 ms based on the binary digits of newly generated private key. Furthermore, this information allows the attacker to reveal the key by observing the ping intervals (100 ms stands for 0; 200 ms stands for 1). We determined the key length is ~343-346 bytes long. Colorado demonstrated that the dynamically recorded ping intervals translate to the key bits, which are repeated in a cycle. The budget of 15000 operations is more than enough for the attacker to

observe the key being created, observe enough ping intervals to recover the bits (< 2800 operations), and then verify the key. Given the evidence above we decided to change our answer to YES after the collaboration."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.7 EMU 6502

A.7.1.3.2.7.1 Question 004 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"Passwords are 50 characters, Authentication uses Spring Framework, Deleting a user requires ROLE_ADMIN

Given this, it is assumed that it is necessary to obtain the administrator password via a side channel in time. No passive observations are allowed.Side note: noticed SecurityContextHolder.setStrategyName(); the javadocs suggest that it should not be set more than once per JVM. This is odd, but we did not have time to investigate implications.

From the LoginView, the password reaches the Spring APIs for authentication, but does not appear to result in observable differences in time. Even if it were, the observation of an admin login is out of scope.User passwords are also stored in User.password in UserService.userRepository, but these do not appear to be observable through active operations."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.7.2 Question 012 (SC Space, Unintended Vulnerable, Answered: Yes)

Response:

"GrammaTech shared data from packet captures indicating that there was an observable space difference per instruction between two different programs, and that it appeared to be mostly deterministic when running the same program. They suggested that the UI for changing emulator registers were probably not functional. We (and they) later confirmed that this was the case -therefore the target user's program execution would be deterministic.The space-based difference is due to Vaadin sending incremental updates, including the register values.

Emulated instructions will appear once per second, in separate TCP sessions (per the supplied benign user script). By performing an offline analysis of all programs, one can build a signature of the space used by the first 25 instructions of each program, bucket them, and use oracle queries to resolve ambiguities (at the cost of using fewer instruction observations).Northeastern performed dynamic testing on 100 of the programs to increase confidence that this was possible. The first 10 instructions were sufficient to distinguish in terms of space. Therefore we believe this will be possible in budget."

Evaluation:

Correct Justification

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.7.3 Question 014 (SC Time, Intended Vulnerable, Answered: No)

Response:

"GrammaTech suggested that the UI for changing emulator registers were probably not functional. We (and they) later confirmed that this was the case -therefore the target user's program execution would be deterministic.

Emulated instructions will appear once per second, in separate TCP sessions (per the supplied benign user script). By performing an offline analysis of all programs, one can build a signature of the timing used by the first 25 instructions of each program, bucket them, and use oracle queries to resolve ambiguities (at the cost of using fewer instruction observations).

Northeastern performed dynamic testing on 100 of the programs to increase confidence that this was possible. However, there was no observable time difference. Therefore we believe this will be NOT be possible in budget."

Evaluation:

Correct Justification after review

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.7.4 Question 016 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Loop Catalog reports monotonic loops within the app. Recursion (Dashboard) was reviewed but nothing of interest within the app itself. No synchronization on monitors or immediate use of synchronized APIs within app. Reviewed Loop Call Graph, no potential vulnerabilities found. We revisited this with other teams during the live engagement, but nothing further was found."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.8 InAndOut 1

A.7.1.3.2.8.1 Question 007 (AC Time, Unintended Vulnerable, Answered: Yes)

Response:

"Two users, one attacker, one benign user.

1)Attacker logs in

a)Submits an order and grabs confirmation number

b)Translate x into a different binary confirmation number (i.e. 1-0-4-8-0-0 into 1-0-1-1-0-0)

c)Submit translated confirmation number in the pick up order page

2)Benign user logs in and submits an order

3)Benign user tries to submit pickup order and is hung for more than 75 seconds

An infinite loop can be caused if the validationIndex is not in the processedOrders and the validation code is in the binaryCodesInUse. This can be caused by supplied a malformed validationIndex such that the orderIsReady check will always be false.The mapping from the validationIndex to the binary code is faulty in that it just translates the binary code above zero to one instead of inverse the original encoding. This is located in ProcessedOrders.java - getBinaryCodeFromValidationCode(String validationCode).

An interesting topic Colorado brought up is that the intended vulnerability could be blocking the queue to the set of orders by the benign user submitting an order and then the attacker sending a bunch of order requests to push the benign user's order towards the end of the queue. This causes the benign user's order to be processed last and thus the time for trying to pick up the pizza could be limited."

Evaluation:

Confirmed their exploit for unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.8.2 Question 027 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

"[…] Since the attacker has control over the order's components and quantity, this means they have control over the values of toggleConnector and initialStage values. This is where fuzzing comes in handy to determine the values (if any) that could be set for the components and number of pizzas.

Colorado performed this fuzzing of the components and the quantity and found a specific exploit. Since there were 15 different toppings and up to 10 number of pizzas this means there are 2^15 *10 choices and only 1 causes this vulnerability. The pie order is 2, 3, 5, 7, 11 for components and 7 quantity."

Evaluation:

Confirmed correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.8.3 Question 049 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"There is a detectable difference in time of the response time from the Inventory Manager to the Server. The process is that the user submits a request to the server and the server sends a request for the Inventory Manager to update in the OrderTaker.java -takeOrder method.

The notifyStockCooking sends the update request to the inventory manager and then the server starts to calculate the validation code (where the vulnerable code is). Then in calculateValidationCode it runs a loop to set the validation code depending on the random binary code. In this loop it takes every bit set in the binary code and sets the validation code to be the binary position representation of that bit (i.e. 101000 = 1-0-4-0-0-0 where the third position is 2^(3-1) = 4). This is generated in the method obtainIndexFromCONTROLLER method which has the timing side channel in that how many times the loop runs depends on the bits set to 1 in the binary code from the input toggleConnector.

After the validation code is generated an update is scheduled for the inventory manager. The Vanderbelt team ran their measurements on the time between the first notify update of the Inventory Manager and the time back from the server to the Inventory Manager. They found a correlation with 64 different timings and thus could predict based on this correlation what the binary number is and then the confirmation number from the encoding."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.8.4 Question 055 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"The size of each packet sent doesn't have any correlation to the confirmation code. This experiment was run by Vanderbilt and they determined that the size of the packet varied over 200 different order requests between the Server and the Inventory Manager (IM). Their tool (Profit) produced a random display of packet size for the different orders given.

We did further analysis to support Vanderbilt's argument. From looking at the relevant code section where the interaction between the Server and IM happen we concluded that the size of the packet is determined by the type of the message. There are 3 different types of messages, updateStock, reverseStock, and recordStock for the IM. The only type that had a different packet size was recordStock because it contained information on the time to be indexed in the orderHistory in the StockConductor. This didn't produce any information on the confirmation number."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.9 InAndOut 2

A.7.1.3.2.9.1 Question 028 (SC Time/Space, Intended Vulnerable, Answered: No)

Response:

"From the collaboration we discovered that there wasn't any correlation in time and a slight correlation in space that didn't fit in the input budget and 70% probability. The Vanderbilt team used the Profit tool to determine this answer and used the two graphs to determine the correlations. The space and time were measured from the Inventory Manager to the Server."

Evaluation:

Incorrect; failed to discover the intended vuln

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3.2.9.2 Question 039 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Two users, one attacker, one benign user.

1)Attacker logs in

a)Submits an order and grabs confirmation number

b)Translate x into a different binary confirmation number (i.e. 1-0-4-8-0-0 into 1-0-1-1-0-0)

c)Submit translated confirmation number in the pick up order page

2)Benign user logs in and submits an order

3)Benign user tries to submit pickup order and is hung for more than 75 seconds"

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.9.3 Question 053 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"Unlike the inandout 1 counterpart the AC Space vulnerability does not have a specific pizza order that causes the vector to fill with integers and cause high memory usage.

From the reverse call graph of the calculateStageSeries method we found that it called the values for intialStage were set by the components and the toggleConnector was set by a large set of binary operations in pullOvenCookingSwitch method. This method takes in the PieOrder which is controlled by the user's request for a valid order.This object contains the which 15 components are on the pizza and how many to make out of 10 quantity which means there are $215*10$different pizza orders.

Since the attacker has control over the order's components and quantity, this means they have control over the values of toggleConnector and initialStage values. This is where fuzzing comes in handy to determine the values (if any) that could be set for the components and number of pizzas. No values for the pizza's components and quantities that cause the memory exploit."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.10 Litmedia 1

A.7.1.3.2.10.1 Question 009 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"Dynamic analysis leads to exploit:

●Run app server on servernuc, and start the memory monitoring script

●Goto login page, login as "sstevens"

●Click on the link to article "The Poet" article (the largest article) or issue GET request to "https://servernuc:8443/media/9792180639" several (50 -60) times in quick succession. This was done by pressing the refresh key on the browser repeatedly.

○The cost per request is no more than 88 bytes including header. So the total cost for the attack should be less than 10 kB, the budget.

●Observe memory monitor to note that memory exceeds 600 MB."

Evaluation:

Confirmed this exploit; unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.10.2 Question 032 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"During the live collaboration, Vanderbilt's experiments with network capture supported our analysis. If one can find away to bucket these somehow then they might be able to do it but from our experiments and evidence provided by Vanderbilt's network analyzer we cannot find any observable difference.

Vanderbilt also pointed out that there are 60 unique possibilities for the preference value. So, technically you may find it using the oracle queries (60 queries needed). But the cost per oracle query is 10 for this question and the operational budget 100. Hence, one can use at most 10 queries. This makes it a lost cause.

The other technicality is if you can somehow bucket the articles using category first and then use oracle queries to pinpoint the exact article. Even this is problematic because

1)We (us and Vanderbilt) were not able to observe any significant timing difference.

2)Assuming our tools did not catch the timing difference, we still need to figure out the tags. There is no restriction on how many tags can be there per article. As we know there are 25 tags, potentially there are 2^25 possibilities. We would need an enormous number of oracle queries to differentiate between them (2^25)*10. Hence, that also doesn't work out.

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.10.3 Question 056 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"We performed dynamic analysis and used Wireshark to determine the size and timing of packets from server to client for the bookmark page.

For 0 bookmarks:

Size of HTML = 3309 bytes, Wireshark size of packet = 3415 bytes

For 1 bookmark, a HTML table row is added to the "bookmark list" HTML page, such as: <tr><td>8527304688</td> <td>A Baby Song</td><td></td></tr>The above adds a linear function of (size of title) to the HTML.

Sizes for 1 bookmark:

Size of HTML = 3400 bytes, Wireshark size of packet = 3506 bytes

The attacker can further use the "Browse" link to get the list of all articles and their title sizes for a cost of 1 operation. The question then boils down to: can the attacker map the (sum of sizes of titles of N bookmarks) to a unique list of bookmarks? […]

Vanderbilt's tool Profit reveals that in the case of maximum 40 bookmarks there is not enough entropy in the information captured to leak the secret. This supports what we discovered. Hence, we stick to our original answer."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.11 Litmedia 2

A.7.1.3.2.11.1 Question 001 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"Taint from the frequency leads to a few branch conditions of which the "updateFavorites" method is most relevant to the question --it compares the frequency of two entries in the client's preference profile and rearranges them (based on a complex logic; not clear what is the goal of the rearrangement). See code below in MediaLibraryImpl.updateFavorites, particularly paths leading to the recursion. The method is interesting for this question because it is recursive, which could lead to differential time consumed based on the algorithm used to rearrange preference entries.

Another potential side channel is the size of the images (ads) sent to the user with an article. We hypothesized this to be a side channel because the ads (intuitively) should be selected based on the preferences of the user. However, we did not find evidence that the preference (frequency of article access) is used to select the ads in the code.

[…] Vanderbilt also pointed out that there are 60 unique possibilities for the preference value. So, technically you may find it using the oracle queries (60 queries needed). But the cost per oracle query is 10 for this question and the operational budget 100. Hence, one can use at most 10 queries. This makes it a lost cause."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.11.2 Question 050 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"Additional background states "The target user will sign in to the application and view the target literature." This means the victim will not be viewing the article in Incognito mode, as by default incognito mode is OFF after login.

There is no observable in the request sent from the victim to the server for an article -the size of the URL is same for all articles: size of "https://servernuc:8443/media/9792180639"For each article viewed by the victim without the incognito mode, the attacker can observe two sets of packets (as observed using Wireshark) from the server to the victim:

●First, the server sends the HTML, which contains the article text, and hence is the relevant observable.

●Second, on rendering the HTML browser requests and the server sends the image (adXY.jpg), which has no relationship with the article being displayed, as the choice of which ad to display is not based on the article requested; rather it is based on a function of a Gaussian random number and the clientId. So this is not a relevant observable.

Evaluation:

Confirmed correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.11.3 Question 058 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"During live collaboration, Vanderbilt provided supporting evidence to our take-home answer. Using their tool, they sampled 300 cases (from the input space for secret) and discovered that about 2.4 bits can leak. Which means to get the remaining 5.8 bits we need about 56-57 oracle queries, which is well within budget.

Team NorthEastern collected the data for bookmark list sizes of 6 and 7. Their data suggests that no bucket is larger than 150. Team Colorado points out that in the worst case when the size of input space for secrets size is 65C7This is a large number = ~696 million possibilities. It is statistically impossible that buckets less than 200 operations will form. In which case, it is very unlikely that the attacker will be able to use the oracle queries to figure out the correct bookmark list within the budgeted number of operations.

Although, the evidence says so otherwise, it is likely that this is due to the fact that the evidence comes from sampling of cases and not entire input space has been analyzed. We do note that for a confident Yes answer, we need to collect data for all possible combinations of bookmarks. None of the teams possess the complete picture for the app. Statistically it looks impossible. Hence, we are changing our Answer from Yes to No, citing lack of evidence."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.12 Phonemaster

A.7.1.3.2.12.1 Question 003 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"Last user takes 2.65 seconds to compute hash and login. First user is 0.75 seconds. Since the list of users is in an ordered list, we can compute the approximate user by calibrating the time it takes for each user to be iterated over on the list and dividing the observed login time by this constant to retrieve the index of the operating user. Northeastern wrote an exploit during the live engagement to confirm our hypothesized attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.13 Powerstate

A.7.1.3.2.13.1 Question 031 (AC Space, Intended Not Vulnerable, Answered: Yes)

Response:

"The vulnerability comes from finding a specific MATPOWER file that causes the data structure Array2DRowRealMatrix to fill up with a lot of doubles. This is caused by the method solve in PowerStateSolver.java file. This method has up to a nesting depth of 4which, with the right parameters, creates a very large Array2DRowRealMatrix and fills memory up to the resource limit. These matrices are then added to another Array2DRowRealMatrix at the end of the solve method which stores onto a set of 10 of these constructed data structures.

These parameters are contained in the space.m file provided with this report and the runspace.sh file that submits the MATPOWER file. This exploit was created by Northeastern using their dynamic analysis tool to generate a MATPOWER file that causes one of these extremely nested loops to fill up the JVMs memory."

Evaluation:

Confirmed exploit; unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.13.2 Question 037 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Vulnerability reported by Northeastern:The vulnerability that is exploited is contained in the method solve in the PowerStateSolver.java class. More specifically the vulnerable code segment is the number of processors running each processing of the MATPOWER file.

The attacker can submit enough requests all running at the same time to fill up the instanceCount variable and cause the number of cores to run out and prevent the benign user from submitting a request. This exploit is contained in the runtime.sh where it submits the time.m file 15 times to fill up the number of cores on the reference platform."

Evaluation:

Confirmed exploit; a bit different than the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.14 PSA

A.7.1.3.2.14.1 Question 010 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"In take home session, our team successfully found the vulnerable code in AnalysisServlet.java and we were able to make a thread sleep forever. Unfortunately, we were not able to have a working exploit to delay the benign request for an hour.

During the live collaboration, we shared our discoveries with others. We found that the benign request is handled in class RootServlet.java. It is simple and straightforward and has nothing to do with thread sleep or lock files. According to this information, we can draw a conclusion that the only way to delay a benign request is to exhaust the thread pool. We came up with a hypothesis that if there is a lock file created in a directory, the thread points to it will sleep there forever. If the number of sleeping threads exceed the max number allowed in thread pool (typically 200 or 300 in size), we will be able to delay the benign request.

The strategy we have is:

1)Send a POST request points to a non-existing directory and file. The server will create that directory with a LOCK.ME file.

2)Wait for couple of seconds. Send 300 GET requests to that directory created in step 1, they will be sleeping and exhaust the thread pool.

3)Wait for couple of seconds, then send the benign request. We should be able to observe a long delay (theoretically forever). Budget calculation:

All sizes measured by WireShark. A POST request takes 327 bytes in tcpdump, a GET request takes 231 bytes. Therefore, the total traffic size is 327+231*300 = 69,627 bytes = 68 kb < 80 kb. So the attack is within budget."

Evaluation:

Confirmed the exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.14.2 Question 030 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"In live engagement, we shared our hypothesis about PyDisassembly.java class with other teams. GrammaTech shared their experience with similar code to show that it seems to be impossible to trigger anything with the json script generated by the app. Colorado then shared their opinion on customized json scripts being uploaded to the server. They were also able to successfully analyze the custom json file with the app. This information inspired everyone in the discussion and we had a strong feeling that we can do something follow this path.

Together with GrammarTech and Colorado, we successfully build a working exploit with a custom json file which has 3200 pairs of "[]" and a "0" in the middle. Analysis to this custom json file will make the server write a 21MB "disassembly.txt" file as we expected. Measured by

GrammaTech, the tcpdump size of this attack is safely within budget. We are all happy with the result and decided to change our answer to YES.

This is a very good example to show that live collaboration works well on questions like this. All the teams had negative answers to this question before the live engagement. Only after combining the static analysis results from our team, dynamic analysis experiences from Colorado, and dynamic analysis tools from GrammaTech, were we able to come up with a concrete YES answer to the question.

Evaluation:

Not the intended vuln; exploit confirmed

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.15 Roulette 1

A.7.1.3.2.15.1 Question 013 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"When Game.wagersOff() runs, it calculates a time to rotate the wheel. The method WheelSlot.rotate() will sleep for the given time, and the time is used to calculate the winning slot. […]

In wagersOff() the he game state is set to BETS_OFF before the clients are notified, so the time WheelSlot.rotate() sleeps is not observable until bets are off. Without network communication space is not observable either.Note: wagersOn(), wagersOff(), placing bets,etc are running in Secretary.scheduledExecutorService, which has a single Thread. This provides mutual exclusion for certain operations on the server."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.15.2 Question 019 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"All IO in the app was reviewed and was either network IO, writes to stdout/stderr, or serialization of the player database (to the file playersDB.ser).

The mapDB serializes Participant instances using com.cyberpointllc.stac.roulette.persist.ParticipantEncoder. Participant.publicEmpty, a String, is serialized. Is the String length bounded? It comes from SenderReceiversPublicEmpty.toString(), and may be based on user input from the connecting client. This in turn depends on SenderReceiversPublicEmpty.face and SenderReceiversPublicEmpty.publicCode. While face is limited to length 25, publicCode is based on BigIntegers which may be derived from user input during login.

During the live engagement, the potential vulnerability was reviewed with the other teams, and nothing was found which would cause publicEmpty to grow in a potentially exploitable way."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.15.3 Question 048 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"In order to determine the bet, at least the bet amount must be observable in space.

he PlaceBet class transmits using a long, which is a fixed length, so the transmission from client to server does not reveal the bet. The status messages from the server back to the client usually encode the amount as a string, but this only reveals at most to the power of 10 what the amount is.

On the server side, the taint of the wagerAmount was reviewed for sinks and influence over conditions; nothing relating to a space-based network observable was identified."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.15.4 Question 054 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Reviewed potential recursion. There are commands for repeating and scripting which appear recursive, but are client-side anyway and guards exist to prevent repeating a repeat. We had discovered the capacity of the BetList.wagers was based on Quantities.capacity, which was being divided by 12 when a DozensWager was placed. We also verified that the BetList.wagers was only constructed once during initialization of the server, which prevented the division in DozensWager from affecting the capacity of the BetList.wagers.

During the live discussion, GrammaTech noted that the fixed capacity of the wagers ArrayBlockingQueue could be a way to block the benign request. However, they also noted that capacity of the queue requires 10,000 wagers to fill the queue, which is beyond the budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.16 Roulette 2

A.7.1.3.2.16.1 Question 011 (SC Space/Time, Intended Vulnerable, Answered: Yes)

Response:

"When Game.wagersOff() runs, it calculates a time to rotate the wheel. The time also depends on the amount of time the app selected for issuing bet reminders during the previous round. spinTheWheel() will sleep for an amount of time which is used to calculate the winning slot.

Game.wagersOn() schedules betting reminders for players for when they have not placed a bet. The interval is randomly selected and is between 10 and 50 seconds. This can be directly observed by (once) placing a bet which exceeds your bankroll, thereafter notifications will arrive until a bet is placed.

In the next round, the betting reminder interval is combined with a random time which is used to determine the winning slot. However, the random time is on the order of 0.005, and therefore the 10 to 50 value from the previous-reminder-interval should dominate (the random time is a rounding error in predicting the winning slot)."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.16.2 Question 033 (SC Space, Intended Vulnerable, Answered: No)

Response:

"Bets are transmitted from client to server using PlaceBet using a long. On the client side this is performed in RouletteUser.situateWager. On the server side it is received at PlayersExchange.handleReceivedMessage(). The server may respond with various status messages, including the validity of the bet (whether it is in range), whether the client wins, etc.

The PlaceBet class transmits using a long, which is a fixed length, so the transmission from client to server does not reveal the bet. The status messages from the server back to the client usually encode the amount as a string, but this only reveals at most to the power of 10 what the amount is. On the server side, the taint of the wagerAmount was reviewed for sinks and influence over conditions; nothing relating to a space-based network observable was identified."

Evaluation:

Incorrect; failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3.2.16.3 Question 046 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"The mapDB serializes Participant instances using com.cyberpointllc.stac.roulette.persist.PlayerDigitizer. Player.publicEmpty, a String, is serialized. Is the String length bounded? It comes from DialogsPublicEmpty.toString(), and may be based on input from the connecting client.

Nothing was found which would cause the file to grow in a potentially exploitable way."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.17 Securgate

A.7.1.3.2.17.1 Question 008 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"Using the packet size for the blocks between the server and the client had different sizes that were consistent with 13 different lengths the attacker could determine which plate came from these 13 different buckets. Furthermore from the analysis of the size of a block could only have at max 8 plates. The suspected vulnerable code is in the Communicator.java class.

The vulnerable line is where the AES encryption is being set to NoPadding which means that there is a direct translation of the plate the the size of the packet. This gives these plates placed in different blocks and the attacker can use the oracle queries to determine the block the plate is in and which plate sent back to the benign user."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.17.2 Question 018 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Team Colorado shared with us the possibility to create an image which can spawn 200 threads within budget. This requires us to set the transparency bit ON. Then you can create an image with 9000x9000 blocks which can then spawn enough threads within the budget. This does cause the program to go beyond 500s.

However, after a discussion among all the teams it is clear that for the exploit to succeed the attacker needs to find a way to upload an image on the server side. From the description, there is no operation which the attacker can use to upload the image in the directory containing license plates. As it is not mentioned anywhere that the attacker has access to the directories on the server side, we do not believe that this attack is in scope.

Hence, we stick to our answer."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.18 Suspicion 1

A.7.1.3.2.18.1 Question 020 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"Using our IO subsystem and Log subsystem we found there are two modes in which the app can write to a file. Two modes to write the file

(1)DistrustSecretary.handleReceivedMessage() -> individual commands

(a)Write a log file (stac.log). No big messages written

(2)MorseCode.decode() -> MorseCode.unzip() -Writes the received message to a file and then reads from the file and attempts to decode it.

This is suspicious as there is no need to store the received message in a file and then read it from the file. Hence, we inspected the code. We note that our Loop Catalog also marks the loop in the following method as suspicious (non-monotonic loop writing to a file). MorseCode.readInt() is a custom method and thus we believe negative bytes can be read i.e. num can be made negative. This will potentially make the loop running for a long time.MorseCode.recordQuantity has a check that foils it.

This check removes the possibility of a large file being written. Also this file is deleted after the message is read. Hence, we believe the vulnerability can not be triggered within budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.18.2 Question 026 (SC Space, Intended Vulnerable, Answered: No)

Response:

"During the live engagement, we captured the network traffic and tried to measure the difference in packet sizes. Based on our measurements, we were not able to confirm the attack as all the packets were of the same size (1056 bytes). While static analysis does not point to a code that will pad the packets to make them of uniform size, the dynamic analysis evidence says otherwise. Hence, we note down that there is a plausible attack and static analysis evidence to support it, but in practice we are not able to observe any difference in terms of size of packets.Thus, we change the answer to No."

Evaluation:

Incorrect; failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3.2.18.3 Question 040 (SC Space/Time, Intended Vulnerable, Answered: No)

Response:

"[…] We first tried to observe if there is any size difference in packets generated by these two commands. Packet capture using wireshark did not reveal anything. Then we hypothesized that maybe due to presence of a loop a timing differential is created and that can help to reveal supplementary information to segregate the captured packets.

Loop Catalog reveals that there are 97 loops in this app, but only 3 are contained in the methods of interest -StartCompetition.execute, DistrustCompetition.generateAndSendObjectives, and DistrustCompetiton.isPasswordCorrect. The loop in the method execute is trivial loop to iterate over list of argumentsThe loop in the method generateAndSendObjectives iterates over list of participants so that it can send messages to everyone.

Initially the loop gives the appearance of a segmented side channel but we notice that the loop is allowed to iterate through all the characters of the Password no matter what. Hence, there is

no timing differential in this loop. We do note that this loop can leak the length of the password but not the password itself. From our audit, we did not find any other code which processes password and can create a time differential.

During the live collaboration, other teams agreed with us. Team Colorado pointed out that there could be a possibility that one can leak the Private key of a benign user and use that to learn the secret. But no supporting evidence for this hypothesis was found. Hence, this possibility was rejected. Due to lack of any additional hypotheses, we stick to our original answer, No."

Evaluation:

Incorrect; failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3.2.19 Suspicion 2

A.7.1.3.2.19.1 Question 022 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"We figured from the description that after executing the command to start the game, the method DoubtTournament.generateAndTransferMissions is called by the game coordinator to send password (along with the location) to players who are not spy.

The only other command that is related to the password is Accuse command to accuse someone of being a spy. But this will end the game and hence is not as interesting. Thus, if at all there is aside channel, it must relate itself to the packets generated by the Start Game command which sends password over the network.

Initially the loop gives the appearance of a segmented side channel but we notice that the loop is allowed to iterate through all the characters of the Password no matter what. Hence, there is

no timing differential in this loop. We note that this loop can leak the length of the password (secret) but not the password itself. From our audit, we did not find any other code which processes password and can create a time differential."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.19.2 Question 044 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"Two modes to write the file(Revealed by subsystem interactions -IO SS and Log SS)

(1)DoubtSecretary.handleReceivedMessage() -> individual commands

(a)Write a log file (stac.log). No big messages written

(2)MorseCode.decrypt() -> MorseCode.expand()

(a)Writes the received message (if the advanced mode is set) to a file and then reads from the file and attempts to decode it.

(b)Following code is suspicious (Loop catalog catches it)

During the live engagement, team GrammaTech showed what input value can exploit this particular code segment. The num must be set to -2,000,000,000. This triggers the code we found during the take home stage and cause it to exceed the resource usage limit. We confirmed their exploit ourselves and hence, we are changing our answer to Yes."

Evaluation:

Confirmed GrammaTech's exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.19.3 Question 057 (Null SC Space, Intended Not Vulnerable, Answered: Yes)

Response:

"MorseCode.encodePointer returns a byte array of length 3 which contains a base-3 number with 0's padded as needed. Since the locations are 26 in number, 3 bytes are sufficient (26 = 222 in base 3). In the method processPointer, these 3 bytes are stored in the first 3 slots. In the method grabMagnitude() these are read one per each turn. Which means for every question asked, the attacker can observe the packet sizes and learn the encoded sitePointer 1 byte at a time. This packet is sent to everyone when the question is answered -> In the method DoubtSecretary.handleReceivedMessage(), if a tournament coordinator receives an answer which is to be forwarded to everyone then he calls the method DoubtTournament.transferAll which in turn calls DoubtTournament.grabMagnitude, thus leaking the secret.

Attacker does not haveto do anything in particular for this attack. He just needs to wait for 3 questions to be asked. 3 questions are asked no matter what happens otherwise the game will not progress further."

Evaluation:

Incorrect; They missed that the round number gets incremented before the first answer message, and assumed that any variation they saw in packet sizes must be due to leaking the secret.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3.2.19.4 Question 060 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"During the live collaboration, team GrammaTech shared the exploit they found. We note that it is same code we found for Question 44 in this app. This code writes to a file in the MorseCode.writeRepeated. More importantly, this code also creates a OutputStream (outStream) to write to a file. The vulnerability for this question lies in the creation of the OutputStream. When you pass a negative number to this loop (the variable num), it causes the loop to iterate for a long time and leads to consumption of large amount of memory for creation of the OutputStream.Team GrammaTech exploit sets the num to -2,000,000,000. This triggers the aforementioned code causes it to exceed the resource usage limit.We confirmed their exploit ourselves and hence, we are changing our answer to Yes."

Evaluation:

Confirmed GrammaTech's unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.20 ThermoMaster

A.7.1.3.2.20.1 Question 021 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"The only code that writes messages to the log occurs in PID.run(),in a loop that can be forced to run a maximum of 32 million times. We can also satisfy ourselves that this will be the case arithmetically: the loop runs 32 million times. Once every 1000 runs, we cache a message, once every 10,000 runs we log those cached messages (and clear the caches, meaning there is not an opportunity to mount a polynomial logging attack on those caches). Each message is 17 or 18

characters long, and are logged as ASCII text (one byte per character). This means approximately 32,000 messages will be logged in a maximal run, with a maximal size of (32,000)(18) = 576,000 bytes, or about 0.5 MB, as we observed from our attack.

In theory, if writing the log file were slow enough in comparison to the CPU, it would also be possible to cause the logger to get backed up, causing multiple "reset log file" messages to be skipped while the logger processes the log messages from an earlier request, followed by logging a large file containing the log messages from multiple requests; however, in practice, the logging of 180 characters worth of log messages takes less than or equal time to running the loop in PID.run() 10,000 times, so there is not a backlog, and what we observe after queuing many instances of the above exploit multiple times in succession is always a file of approximately 0.5 MB."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.20.2 Question 038 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"Retrieving the setpoint via accessing the User's PID can be dismissed by investigation the code that retrieves it; the user's 15-20 character password is required to gain access to their User object and the PIDs contained within. The fields used to store the setpoint, however, are more interesting, as they are used by both the benign user's requests and the attacker's requests, leaving potential for the attacker to see some afterimage of a previous request, if everything is not cleared correctly.

The same is true of the fields Server.pid and Server.user, which likewise hold the secret indirectly: if they are set, they are also reset to null before the return from Server.serve().Thus, after the benign user's request that sets the setpoint secret, all the fields that hold it are reset, except for the list of users.

A user can only be retrieved from that list by sending the user's name and a password that MD5 hashes to the same value as that user's original password, which, although sub-standard for protecting against account theft, is sufficient to prevent an attacker from reliably guessing an account name and password within the given budget. Thus, all the places that the secret is stored are sufficiently protected against an attacker with the given operations and budget, so there is not a corresponding side channel vulnerability."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.20.3 Question 052 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"This function contains a loop that is capped to run atmost 32 million times, and contains costly functionality, including logging and the mathematical calculations for simulating the behavior of a PID controller. If the temperature is not initially incredibly close to the target temperature, the loop runs until it is close, until the current temperature diverges to positive or negative infinity

(which is easily possible due to a bug in the PID's derivative calculation), or until it reaches the cap of 32 million iterations.

M

Mobspasswordpass

P

disable proportional

disable integral

disable derivative

disable windup

set current

999.99

set setpoint

-999.99

The above request disables all temperature control measures taken by the PID controller, and then sets the current temperature far enough from the target set point that it is incredibly unlikely that the current temperature will reach the set point and terminate the loop before it has reached all 32,000 iterations.

During the engagement, team Vanderbilt and team Northeastern demonstrated their exploits. These exploits are similar in nature to the one we have included. The idea still remains the same, make the vulnerable loop run for more than 15 seconds and deny the service to the benign request. To do that we still need disable all control measures but the setpoint temperature must be set to -1.175494351e-38 and the current temperature must be set to 1.175494351e-38."

Evaluation:

Confirmed exploit; different from the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.21 Wordshare

A.7.1.3.2.21.1 Question 002 (SC Space, Unintended Vulnerable, Answered: No)

Response:

"It is possible that attacker can observe the size of the response packet generated by the server which can reveal the secret word. The "findAndEncrypt" function encrypts the plain text of the secret word. We can see in the above PCG (Projected Control Flow graph), if the secret word size is more than 16 bytes it throws an exception, therefore the user cannot send a secret word more than 16 bytes long. […]

We conclude that the attacker cannot reveal the plain text of the secret word."

Evaluation:

Was previously correct but another team found a vuln.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3.2.21.2 Question 005 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

"It is possible to craft an input during the "add" document operation by sending the file name with many aliases, where each alias is separated by colon (:), which will cause the server to create that many number of clones by writing the data of the original file. An exploit is included with this report. The server splits the document name based on colon (:). If we add document with aliases and after adding this document, if we send a request to add the document with these aliases (excluding previous document name) again, then the server will copy that document to all these alises.

As noted next to the final answer, the vulnerability described is not in scope because it writes multiple files, and the question calls for one."

Evaluation:

Originally had Yes but changed their answer to No after the live portion. Determined to be 'out-of-scope' due to the multiple files. Was originally deemed correct but another team found a vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.3.2.21.3 Question 047 (SC Time, Unintended Non-Vulnerable, Answered: No)

Response:

"The Attacker uses a random secret key to send a search request to the server in order to get the documents containing that word. The server first checks whether user is privileged user or

not. The server then calculates MAC using the attacker's secret key and benign secret key and performs Arrays.equal in order to check whether the MAC calculated from the attacker's secret key is equal to the benign secret key or not. Arrays.equal performs a pairwise checking in the loop. If 1st pair is not equal it returns quickly, if 1st pair matches but 2nd pair doesn't then it would take little more time than previous and so on.

So, there is a potential segmented side channel to get the actual secret key which can be then used to decrypt the secret. But the time difference is very small and is not observable and we don't believe it is possible to get the secret key within budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.22 YapMaster

A.7.1.3.2.22.1 Question 015 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"The vulnerability is contained in the get method of MemoryMapReader. The vulnerability was found by the Colorado team from testing the get method several times to find that it increases the file size.

The FileChannel fc is used to read from the file yet it uses “rw” for the random access file and thus has permissions to write. In this case the writing occurs when fc.map(READ_ONLY, start, bufferSize) in which it tries to read from a high number (i.e. 900000) the buffer size which the map function fills the file before trying to do the read if it is empty. The get request allows you to supply an id specified by the request without preventing this check and this causes the file to be generated of being size 115 MB.”

Evaluation:

Confirmed correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.22.2 Question 017 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“We could not find any observable difference in time that would lead the attacker to guess the most recent yap of another yap alias and to get the decrypted message. […]

Posting YAP operation is not dependent on the content of posted YAP message. Therefore, any timing difference is observed will not allow the attacker to guess the content of the encrypted message. The only timing difference that can be observed is due to finding proper location in cache to store the posted/updated yap. However, the observed timing different is independent on the content of the posted YAP.

During the live engagement GrammaTech demonstrated that there is a way to overwrite the messages in cache of a benign user. However, there is no way to read the contents of the cache of a benign user. Vanderbilt suggested to see if the network traffic reveals something. The network capture also does not reveal any significant time or space difference.

GrammaTech brought up the possibility that maybe there is a way to use the fact that one can overwrite the messages in the cache of a benign user to reveal the PrivateKey of that user. From a cursory investigation of the cryptographic component of the app, it does not appear to be the case. Hence, we stick to our take home stage answer of No.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.3.2.22.3 Question 042 (SC Space, Intended Vulnerable, Answered: No)

Response:

“We could not find any observable difference in space that would lead the attacker to guess the most recent yap of another yap alias and to get the decrypted message. […]

During the live engagement GrammaTech demonstrated that there is a way to overwrite the messages in cache of a benign user. However, there is no way to read the contents of the cache of a benign user. Vanderbilt suggested to see if the network traffic reveals something. The network capture also does not reveal any significant time or space difference.

GrammaTech brought up the possibility that maybe there is a way to use the fact that one can overwrite the messages in the cache of a benign user to reveal the PrivateKey of that user. From

a cursory investigation of the cryptographic component of the app, it does not appear to be the case. Hence, we stick to our take home stage answer of No."

Evaluation:

Incorrect; they failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4 Northeastern Specific Responses

A.7.1.4.1 Answer Statistics

Table A-149: Engagement 7 NEU Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	11	6	55
SC Time	13	12	92
SC Space/ Time	7	5	71
AC in Space	18	16	89
AC in Time	11	11	100
Total	**60**	**50**	**83**

Table A-150: Engagement 7 NEU Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	31	22	71
Not Vulnerable	29	28	97
Yes Answer	23	22	96
No Answer	37	28	76

A.7.1.4.2 Northeastern Specific Responses

A.7.1.4.2.1 Calculator 3

A.7.1.4.2.1.1 Question 023 (AC Space, Unintended Vulnerable, Answered: Yes)

Response:

"The division operation invokes LargeNumeral.subtract which uses memory-expensive operations(e.g., recording carries using a list). By submitting the expression generated by the python expression "+".join(["10*499/2" * 250]), the server's memory usage can be brought above the 4 GBlimit."

Evaluation:

Confirmed this exploit; unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.1.2 Question 025 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"Our tools and analysis do not indicate the presence of an AC space vulnerability."

Evaluation:

Correct but minimal justification.

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.1.3 Question 034 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Our combination of static and dynamic analysis revealed a potential AC Time vulnerability in the LargeNumeral.divide method. We also came to this conclusion for Question 45 in Calculator 4. After collaborating with other teams, where we compared exploits and used a combination of static analysis tools and profiling the challenge in VisualVM, we arrived at a consensus that the implementation of the divide method in this challenge causes the AC Time vulnerability in calculator_3."

Evaluation:

Confirmed that the divide exploit is valid

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.2 Calculator 4

A.7.1.4.2.2.1 Question 036 (AC Space, Intended Vulnerable, Answered: No)

Response:

"Similarly to question 43, we found a working exploit but it was out of budget."

Evaluation:

Incorrect conclusion with minimal justification; they did not find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4.2.2.2 Question 043 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Our initial analysis demonstrated that an adversary could slow down methods in the challenge's divide method. We constructed an expression that exploits this behavior in calculator that exceeds the time budget by repeatedly performing division on successively larger numbers. During the collaboration we found our initial exploit exceeded the adversary's resource budget, and a reduced exploit did not exceed the time budget for a successful exploit. By collaborating with other teams and manually analyzing the challenge we came to the conclusion that there is no exploitable expression in calculator_4."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.2.3 Question 045 (AC Space, Intended Vulnerable, Answered: No)

Response:

"Our tools and analysis do not indicate the presence of an AC space vulnerability in this challenge."

Evaluation:

Incorrect conclusion with minimal justification; they did not find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4.2.3 CyberWallet 1

A.7.1.4.2.3.1 Question 006 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"We analyzed the program allocations and we believe the challenge does not contain any intended AC Space vulnerability."

Evaluation:

Correct, but very minimal justification.

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.3.2 Question 035 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"The account page loads an advertisement image into the page. The image is chosen by selecting a picker that strongly depends from the account balance (picker = log10(balance/100000+ small randomness)). Grammatech and Colorado observed that the possible selected images have significantly different sizes, allowing an attacker to infer that loaded image (and then the account tier) by monitoring the response size."

Evaluation:

Correct through collaboration.

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.3.3 Question 051 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"There is check in VerificationManager that prevents the same attack described for Q_059. Such check prevents calling the vulnerable function the destination account does not exist. We did not any other observable side-channel and we believe the challenge does not contain any intended vulnerability."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.4 CyberWallet 2

A.7.1.4.2.4.1 Question 024 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"Two differences in the challenge make the attack described in Question 035 impossible. First, the account balance is multiplied for a random number (instead of being summed), significantly altering the produced result in an unpredictable way. Second, adding +4 in the picker computation shifts the image selection. As a result the page loads images with index between 4 and 9, which we observed do not differ enough in size to allow a reliable attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.4.2 Question 059 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"To steal money from another user, we need his account number and an authorization number. With this information we can perform a transaction with negative value, i.e., moving money from the target account to ours. Grammatech identified a side-channel time in findAccountFace that allows to brute force an account number one digit at the time (10 digits, 100 trials needed).Authorization Number is value between 0 and 100, which can be easily guessed. With around 200active operations we could obtain the information needed to steal money from another account and perform the transaction."

Evaluation:

Confirmed GrammaTech's exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.5 Door Master

A.7.1.4.2.5.1 Question 029 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"We found an infinite loop that makes calls to keyFobs/auth endpoint and hangs forever when the remote server is down or when an attacker sends lots of requests that create and delete keys simultaneously. However, although this can successfully stall a thread, we believe it is no possible to repeat this attack to stall all the web-server threads within the budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.5.2 Question 041 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"MessageLoop class has a method run, if the attacker can observe the traffic he can understand when the first key is generated by looking at the frequency of ping messages sent between the

MessageLoop and the remote socket server. This happens because until the first key is not generated yet message loop pings the server each second. After the first key is generated, the key bits directly affect the sleep time between pings. We could reliably recover the private key by monitoring times between pings."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.6 EMU 6502

A.7.1.4.2.6.1 Question 004 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"We analyzed the program, and it does not seem to contain any timing side-channel vulnerability that would allow any leak of admin credentials."

Evaluation:

Correct, but with minimal justification

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.6.2 Question 012 (SC Space, Unintended Vulnerable, Answered: Yes)

Response:

"We manually observed the size of each TCP connection and analyzed the content for each connection. The attack works in the following way: The attacker can observe sizes of all server responses after the user has clicked the "execute" button. Each server response corresponds to how the page content must be updated, for example, which registers have value changes. Because different assembly programs behave differently at every step, the sizes of responses are different, too. Therefore, the attacker can generate a fingerprint using first N observed sizes of server responses for each file, and then use those fingerprints to discover which source file the user is executing.

We manually measured response sizes for the first 100 source files (out1.s to out100.s), and found out that when N >= 10, the fingerprints are unique among the 100 files. We believe that N is less than 15 for the entire set of 500 source files, which means the attack can be conducted within the budget."

Evaluation:

Correct justification after review

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.6.3 Question 014 (SC Time, Intended Vulnerable, Answered: No)

Response:

"Our static analysis identified time-consuming operations (including a call to sleep(1000), exceptions raised under certain conditions, and several database operations), but none of those operations matter for answering this question. We were assuming that when emulating branching instructions, `BranchExceptions` will be thrown, and subsequently, the server will load the code

from the database again. Then we should be able to leverage the execution properties of the first 25instructions of a source file and fingerprint the consumed time of each instruction executed -- Ideally, for every single branch instruction, the emulator will spend a little bit more time in execution, which will be reflected in the timespan of each connection. We manually measured response timespans for the first 25 connections after user has clicked the "execute" button for the first 100source files (out1.s to out100.s), and we _cannot_ find any obvious increase of the timespan when executing a branch instruction compared to executing a non-branching instruction. This might be because of the variance: We observed a 0.02 variance in the timespan of each connection, which may have disguised the extra time that the branch instruction simulation takes. Although we are inclined to answer Yes to this question, we end up answer No as we cannot form a working attack. We shared this finding with other teams and they agreed with our finding."

Evaluation:

Correct Justification after review

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.6.4 Question 016 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"We analyzed the code and did not find any way to stall the execution for a long time. The only possibility that we have found (actually, we've been told about this by other teams during collaboration) that might stall the execution is by exhausting the connection pool that Spring Framework has. By default this connection pool has a upper limit of 200. However, it seems impossible to create 200 different HTTPS connections within the allowed budget (5KB), which averages to 25 bytes per connection. Therefore, we believe the answer to this question is a No."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.7 InAndOut 1

A.7.1.4.2.7.1 Question 007 (AC Time, Unintended Vulnerable, Answered: Yes)

Response:

"The application contains a vulnerability where an adversary can refer to counterfeit orders in the system. This allows an adversary to bypass a check against invalid orders and force a loop to iterate indefinitely."

Evaluation:

Confirmed potential unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.7.2 Question 027 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

"Profiling the performance of individual methods in the application we observed you could construct a pathological order that achieves an AC Space attack."

Evaluation:

Correct but very minimal justifications; no POC for this application provided. Assuming they collaborated with other teams but did not have time to elaborate on their justification.

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.7.3 Question 049 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"By collaborating with GrammaTech we found a side-channel in time in the calculateValidationCode method that allows an adversary to obtain enough information to obtain the victim's pickup request."

Evaluation:

Validated GrammaTech's exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.7.4 Question 055 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"Vanderbilt's Profit tool indicates no relationship between the size of the requests and the size of the packets."

Evaluation:

Correct thanks to collaboration; no discussion of own tools or findings

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.8 InAndOut 2

A.7.1.4.2.8.1 Question 028 (SC Time/Space, Intended Vulnerable, Answered: No)

Response:

"We could find no evidence of a side channel in either time or space. Dynamic analysis provided by Vanderbilt's tools found no correlation between the content of the sensitive information and the timing of network packets. This analysis did find a correlation between the secret and the size of response packets, but we concluded it was not strong enough to be exploited within the stated budget."

Evaluation:

Incorrect; concluded that the correlation they found could not be exploited within the budget.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4.2.8.2 Question 039 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"The application contains a vulnerability where an adversary can refer to counterfeit orders in the system. This allows an adversary to bypass a check against invalid orders and force a loop to iterate indefinitely."

Evaluation:

Confirmed this is valid exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.8.3 Question 053 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"Static analysis identified loops that could make the application potentially vulnerable to AC Space attacks. Further analysis revealed that return values of specific methods bounded the number of iterations of these loops. By profiling these methods we found the loop bounds are limited to a constant value."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.9 Litmedia 1

A.7.1.4.2.9.1 Question 009 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"Our dependency analysis suggested that the size of the bookmark list is unbounded, and there is a constraint on the size of the bookmark list, which is `bookmarkList.size() == 40`. We then took a look at the code, and we concluded that since this is a multithreading application, we can bypass the size constraint on bookmark list by adding the 40th bookmark from multiple threads. Hence the maximum number of bookmarks is only bounded by the total number of articles in this application. However, we do realize that this is not enough for the application to exceed the memory limit of 600 MB.

The Iowa team suggested a different attack. By accessing the article browsing page from multiple clients at the same time, we can easily push the memory usage of the application to over 600 MB. This is found by human reasoning."

Evaluation:

Confirmed Iowa's potential unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.9.2 Question 032 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"If we only need to discover the top-ranking "category" of a target user's preference, we can simply query the oracle 10 times to leak 11 bits of data and determine the top-ranking category with a90% probability. This was our original intended way of attack. However, during the collaboration, we were informed that we need to leak both "category" and "tags", and we believe that there is no

way to get enough information leak from the Picks list to determine the content in the top-rankingPickNode. We noticed that updatePicks() method implements a sorting algorithm and we tried to determine if the sorting algorithm is a) running in a constant time, and b) its execution time is dependent on the content of PickNode. Our automated analysis could not determine if it is running in a constant time or not (in fact, the analysis believes the execution time is not

constant, but it contradicts the result of our manual measurement). There is also no dependence between the function and the “category” and “tags” of the top-ranking PickNode instance. So in the end, we think the answer to this question should be a No.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.9.3 Question 056 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

“The maximum number of bookmarks allowed in this challenge is 40. We believe it is impossible to conduct the attack since the total number of possible bookmark lists is (65! / (40! * (65 - 40)!),which, after bucketization, will definitely create a bucket with more than 200 elements. This means we will not be able to differentiate between elements within that bucket within the allowed budget(200 oracle queries). Hence our answer to this question is No.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.10 Litmedia 2

A.7.1.4.2.10.1 Question 001 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“Same as the answer for Question 032. If we only need to discover the top-ranking “category” of atarget user’s preference, we can simply query the oracle 10 times to leak 11 bits of data anddetermine the top-ranking category with a 90% probability. This was our original intended way ofattack. However, during the collaboration, we were informed that we need to leak both “category”and “tags”, and we believe that there is no way to get enough information leak from the Picks list todetermine the content in the top-ranking PickNode.We noticed that updatePicks() method implements a sorting algorithm and we tried todetermine if the sorting algorithm is a) running in a constant time, and b) its execution time isdependent on the content of PickNode. Our automated analysis could not determine if it is running in a constant time or not (in fact, the analysis believes the execution time is not constant, but it contradicts the result of our manual measurement). There is also no dependence between the function and the “category” and “tags” of the top-ranking PickNode instance. So in the end, we think the answer to this question should be a No.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.10.2 Question 050 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

We ran dynamic analysis on this with different articles browsed (without retrieving the picture first),and we only found a very small variation (around 1 ms) in terms of the time cost. This

variation also did not show any correlation with the articles browsed. We then manually analyze the source code to see if the ad picture has anything to do with the browsed article. Unfortunately, the picture ID is determined by the product of the hash of user ID and a random number, which means the chosen picture has no correlation with the article being browsed. Hence we believe that the answer to this question is a No.

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.10.3 Question 058 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"The content of the bookmarks page includes titles of all currently bookmarked articles. We measured the size of the bookmarks page (in plain text) with different bookmark settings, and we noticed in the majority cases, it is possible to map to a single set of bookmarks from the size of the page. We then measured the size of TCP streams on the wire considering impact of SSL encryption and compression, and we found out that we can still identify a set of bookmarks from the size of the response packets. However, with the maximum bookmark list size limited to 7 (it is a bad limit that can be bypassed, but we stay with it for now), there are way too many possible selections of different bookmarks: The number of selections is around 65 million, which means we will definitely end up with a bucket with more than 200 elements. Hence we believe the answer of this question is a No. Collecting traffic sizes took a lot of time for us. For some reason the server responded slowly(taking multiple seconds) to our script at times."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.11 Phonemaster

A.7.1.4.2.11.1 Question 003 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"Users are stored sequentially in a List. Therefore, the time required to fetch a user depends on its position in the list. The difference in time is clearly observable observable because there is a sleep in the CheckLogin function. Hence, an attacker can first get the users list then perform another active operation to estimate the time required to scan one more user in the list and finally passively monitor the request of a target user and identify its position in the list by observing the response time. In case of imprecision in the time analysis, the attacker can spend the remaining budget to query the oracle for nearby users."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.12 Powerstate

A.7.1.4.2.12.1 Question 031 (AC Space, Intended Not Vulnerable, Answered: Yes)

Response:

"By sending roughly 900 pq buses with one pv bus and one slack bus, using a linear chain of one-way bridges to feed power throughout the whole system, and omitting two fields (possibly by changing to version 1 of the format), we could get the input file size down to 48K and make the challenge use up > 2GB of RAM."

Evaluation:

Confirmed this exploit for unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.12.2 Question 037 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"We found two ways to trigger the vulnerability: First, randomly generating files very quickly make sone take a nearly (or possibly actually) infinite time to compute, allowing then to just request that file until all the cores are filled up. The other solution is to send failing input which crashes the threads, but the program does not seem to actually account for those threads going away, so you quickly eat up all the cores."

Evaluation:

Confirmed the exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.13 PSA

A.7.1.4.2.13.1 Question 010 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"By either instructing the server to load the deserializer module or an invalid module, the analysis will throw an exception, leaving a lockfile (LOCK.ME) present on storage. Subsequent GET requests for the analysis will then hang. By submitting ~200 requests, the thread pool used to back servlets handling new requests can be exhausted, causing subsequent benign requests to hang. The attacker must interact with the server using HTTP/1.1 over TLS1-1.2 which introduces significant overhead on the wire. By using TLS session resumption, turning off SNI, and specifying exact ciphersuite/signature/elliptic curve parameters, the attack can be brought to ~71 KB, within the attacker's budget."

Evaluation:

Confirmed

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.13.2 Question 030 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"The attacker can upload arbitrary JSON files to the server for analysis. Prior to well-formedness checks, the uploaded JSON will be pretty-printed by the server using GSON (sic). By crafting a malicious JSON payload, the pretty-printing process will expand the uploaded JSON into a form that exceeds the storage limit for this question. As an example, uploading a JSON file with many nested arrays (e.g., 3,200 pairs of square brackets) will exceed the acceptable storage limit."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.14 Roulette 1

A.7.1.4.2.14.1 Question 013 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"We used static dependency analysis to determine how the winning number is generated and the relation between the winning number and the random numbers. The spin time of the wheel is decided by the takeTime() function in the Game class. We analyzed takeTime(), and it's obvious that it returns a random number. We concluded that because the random numbers are cryptographically secure, there is no way for us to predict the random number sequence from the observed time differences between any packets. Also, we believe the space side-channel indifferent packets do not leak anything about the winner number. Hence the answer for this question is No."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.14.2 Question 019 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"We performed static analysis on this challenge and we believe that the only possible way to write any data onto the storage is via the players database (which is a MapDB instance). It seems to us that the challenge will only update the content in the database (instead of inserting new records),and the only controllable field is the user name. However, the user name can have at most 25characters, so it is not a viable way to insert too many bytes into the database. In addition, we are not aware of any vulnerability that will cause MapDB to overflow the storage. Hence we believe the answer to this question is No."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.14.3 Question 048 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"We used static dependency analysis to analyze how the UserBet object is constructed and serialized, and we also manually observed the packets that are sent out from the user to the server

after the user makes a bet. No extra information about the bet is put into the serialized packet, and not enough variations can be observed on the wire from the size of those packets to reveal the bet type, value, and amount. So we believe the answer to this question is No."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.14.4 Question 054 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"We did not find any possible ways to stall the bankroll command of another user."

Evaluation:

Correct but minimal justification

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.15 Roulette 2

A.7.1.4.2.15.1 Question 011 (SC Space/Time, Intended Vulnerable, Answered: Yes)

Response:

"We used static dependency analysis to determine how the winning number is generated and the relation between the winning number and the random numbers. The spin time of the wheel is decided by the takeTime()function in the Game class. We analyzed takeTime(), and it's obvious that it returns the sum of`lastTime` and a random number. We used to believe that because the random numbers are cryptographically secure, there is no way for us to predict the random number sequence from the observed time differences between any packets. However, during collaboration, other teams found out that `lastTime` is big, and the random number is so small (usually between -0.1 and 0.1) that it can be ignored. Dependency analysis further shows us that `lastTime` is determined by `wagerReminderTime`, which is the delay of the bet reminder packet. Hence we are able to predict the spin time of the wheel by observing the delay of the bet reminder packet (which can be identified by its unique size) sent out on the wire. Therefore, the answer to this question is Yes."

Evaluation:

Correct; sounds close enough to the intended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.15.2 Question 033 (SC Space, Intended Vulnerable, Answered: No)

Response:

"We used static dependency analysis to analyze how the UserBet object is constructed and serialized, and we also manually observed the packets that are sent out from the user to the server after the user makes a bet. No extra information about the bet is put into the serialized packet, and not enough variations can be observed on the wire from the size of those packets to reveal the bet type, value, and amount. So we believe the answer to this question is No."

Evaluation:

Incorrect; failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4.2.15.3 Question 046 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"We performed static analysis on this challenge and we believe that the only possible way to write any data onto the storage is via the players database (which is a MapDB instance). It seems to us that the challenge will only update the content in the database (instead of inserting new records), and the only controllable field is the user name. However, the user name can have at most 25characters, so it is not a viable way to insert too many bytes into the database. In addition, we are not aware of any vulnerability that will cause MapDB to overflow the storage. Hence we believe the answer to this question is No."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.16 Securgate

A.7.1.4.2.16.1 Question 008 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"An attacker can leak the plate by monitoring the responses of the /get_image_block requests. In fact, the number of requests depends on the plate image size (in terms of number of blocks).Colorado measured the number of blocks for each image, showing that there are enough distinct buckets to identify the correct image within the budget (passive operations + oracle queries).In addition, the Communicator class uses no padding during encryption keeping a one-to-one mapping between image block lengths and encrypted block lengths.'

Evaluation:

Confirmed

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.16.2 Question 018 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Colorado identified a successful attack that creates an image with a very large number of blocks(10,000). The trick to do this without significantly increasing the image size is to set the image transparence bit, which significantly affects the number of generated blocks. Such image can be used to exhaust all the server threads. However the attacker does not have the capability to upload a new plate image, therefore we concluded that this attack is out of scope."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.17 Suspicion 1

A.7.1.4.2.17.1 Question 020 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

“In `com.cyberpointllc.stac.transput.MorseCode.expand()` function, if the attacker can find a way to bypass the check on `aggregateContent` variable, he will be able to write a large file onto the disk. However, we manually analyzed this function, and we concluded that there is no way to bypass the size check. Therefore the answer to this question should be a No. Our fuzzer did not find anything worth reporting regarding this question.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.17.2 Question 026 (SC Space, Intended Vulnerable, Answered: No)

Response:

“Iowa State's Graph Analysis tools allow an analyst to observe that padding is added to the location pointer. The amount of padding is based on the value of the location pointer. This introduces a side-channel in space that allows an adversary to learn the value of the location pointer based on the length of the packet. However, by dynamically executing the challenge, we could not observe and measure a significant difference in the packet lengths.”

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4.2.17.3 Question 040 (SC Space/Time, Intended Vulnerable, Answered: No)

Response:

“The size of the password seems too small, and the number of possible strings combined with the weak signal given by the size of response packets suggests there is not a strong enough side channel in time for an adversary to leak the password. Static analysis revealed you can leak the length of a user's password, but this is not enough information to recover the full text.”

Evaluation:

Incorrect; they concluded that the side channel was not strong enough

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4.2.18 Suspicion 2

A.7.1.4.2.18.1 Question 022 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

“A side-channel in time in the challenge's decryption methods could allow an adversary to leak the server's private key. These methods may exhibit slow behavior due to the inefficient implementation of arithmetic in the BigDecimal class. During the collaboration we performed

manual analysis over these routines and found there is not a strong enough signal to recover the private key."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.18.2 Question 044 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"In com.cyberpointllc.stac.transput.MorseCode.expand() function, if the attacker can forge packets that are sent on the wire, it is possible to cause a memory explosion by sending a large value for num in packets. Note that the only check that needs to be bypassed is if((aggregateContent += num) > 1000000) { throw new IOException("...") }.It can be bypassed by passing in a large negative value in the first packet: both aggregateContent and num are ints, and they can be negative numbers."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.18.3 Question 057 (Null SC Space, Intended Not Vulnerable, Answered: Yes)

Response:

"Iowa State's Graph Analysis tools revealed a variant of the side channel in space present inQuestion 26."

Evaluation:

Incorrect; They missed that the round number gets incremented before the first answer message, and assumed that any variation they saw in packet sizes must be due to leaking the secret.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4.2.18.4 Question 060 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"In com.cyberpointllc.stac.transput.MorseCode.expand() function, if the attacker can forge packets that are sent on the wire, it is possible to cause a memory explosion by sending a large value for `num` in packets.

Note that the only check that needs to be bypassed is `if ((aggregateContent += num) > 1000000) {throw new IOException("...") }`. It can be bypassed by passing in a large negative value in the first packet: both `aggregateContent` and `num` are ints, and they can be negative numbers. The answer to this question should be a Yes. Ideally our fuzzer should have found this issue _if_ we were correctly modeling `MorseCode.readInt()`. Unfortunately we weren't modeling the function correctly, as a result, `readInt()` was only returning non-negative values during fuzzing. After modeling the `readInt()`function correctly, we are able to find this issue automatically."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.19 ThermoMaster

A.7.1.4.2.19.1 Question 021 (AC Space (Disk), Intended Not Vulnerable, Answered: No)
Response:

"We analyzed the file operations. With the number of loop iterations done during the simulation being fixed and only fixed-length lines being appended to the log file every 1000th iteration the total log size does not seem to depend on any user input. We believe the challenge does not contain an intended vulnerability."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.19.2 Question 038 (SC Time, Intended Not Vulnerable, Answered: No)
Response:

"We could not find a path in the program where an adversary could influence the value of another user's PID setpoint."

Evaluation:

Correct, but minimal justification.

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.19.3 Question 052 (AC Time, Intended Vulnerable, Answered: Yes)
Response:

"Simulation thread joined in a synchronized method immediately after started which locks others out during simulation, in effect making the requests sequential. With the sample taking 10 seconds to compute and 15 seconds of computation needed to successfully attack, an attacker needs a 50%slowdown. We could increase the computation time of roughly a factor of 2 by choosing two values very close to 0 (one positive and one negative) as well as disabling some of the provided options. The repeated calculations involving very small floating point numbers produced a significant slow down in the computation. Specifically, setting setpoint to -1.175494351e-38 and current to 1.175494351e-38 and disabling proportional, derivative, and integral we made the computation slow down to more than 20 seconds."

Evaluation:

Confirmed provided POC when combined with collab

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.20 Wordshare

A.7.1.4.2.20.1 Question 002 (SC Space, Unintended Vulnerable, Answered: No)
Response:

"Our static analysis tool highlighted some very suspicious methods related to finding the prefixes in the trie data structure. However, we could not observe any reliable side-channel in time."

Evaluation:

Was originally correct but another team found an unintended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.4.2.20.2 Question 005 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

"We analyzed the program allocations and we believe the challenge does not contain any intended AC Space vulnerability."

Evaluation:

They failed to identify the intended vulnerability and missed the YES answers from other teams; Justification really insufficient

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.20.3 Question 047 (SC Time, Unintended Non-Vulnerable, Answered: No)

Response:

"We observed the server response times and did not notice any significant difference that can be leveraged to carry on a side-channel attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.21 YapMaster

A.7.1.4.2.21.1 Question 015 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"During read operations MemoryMapReader fills up the target file up to start_id(fc.map(MapMode.READ_ONLY, (long) start, bufferSize)), which is computed from the id chosen by the user (int start = (id - 1) * 128). By simply requesting a read operation and selecting the maximum id accepted by the program (i.e., 9000) we could make the program store a file larger than 100MB."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.21.2 Question 017 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"We analyzed benign network traffic and found that users interact with the service via HTTP. Some requests contain user ids and ciphertexts. An adversary could possibly replay this traffic to profile the decryption routines implemented in the challenge and detect a side channel. With just 1200 operations in the adversarial budget, it seems unlikely an adversary could recover the user's privatekey."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.4.2.21.3 Question 042 (SC Space, Intended Vulnerable, Answered: No)
Response:

"We used static analysis tools across different teams to approximate loop complexity and reason about the different execution times of branches that occur based on the content of sensitive secrets. The teams' analysis allowed us to conclude that yapmaster does not contain aside-channel."

Evaluation:

Incorrect; they failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.5 Vanderbilt

A.7.1.5.1 Answer Statistics

Table A-151: Engagement 7 Vanderbilt Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	11	6	55
SC Time	13	12	92
SC Space/ Time	7	5	71
AC in Space	18	17	94
AC in Time	11	11	100
Total	**60**	**51**	**85**

Table A-152: Engagement 7 Vanderbilt Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	31	23	74
Not Vulnerable	29	28	97
Yes Answer	24	23	96
No Answer	36	28	78

A.7.1.5.2 Vanderbilt Specific Responses

A.7.1.5.2.1 *Calculator 3*

A.7.1.5.2.1.1 Question 023 (AC Space, Unintended Vulnerable, Answered: Yes)
Response:

“Kelinci generated several inputs, but none of them are interesting enough to be investigated further. During collaboration, GrammarTech said they discover an input manually. The input uses the subtract method which contains a complicated nested loop. The exploit is an input that does a lot of subtraction of the form (10^499)-(10^499)+(10^499)-(10^499).... The loop will keep the created array in memory without being gargbage collected.”

Evaluation:

Confirmed the GrammaTech exploit for potential unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.1.2 Question 025 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

“Kelinci generated several inputs, but none of them are interesting enough to be investigated further. We wrote the KelinciWCA's driver for the method handleExpression in the Calculator class. We experimented with the grammars for basic calculator and roman numerals. There were a number of discussions on this during collaboration, but we couldn't find out the exploit by any way.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.1.3 Question 034 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

“Kelinci generated several inputs, but none of them are interesting enough to be investigated further. Colorado came up with the following exploit by using theSingularity fuzzer: (X^LD/(X^LD/(X^LD/(X^LD/..... /III^D)))). Justification was discussed during live collaboration. During collaboration, profiling tool was used to investigate performance bottleneck. The reason could be that is that the '^' and '/' operators are expensive and the calculation involve big numbers which make the calculator to slow down.”

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.2 *Calculator 4*

A.7.1.5.2.2.1 Question 036 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

“KelinciWCA generated several inputs, but we didn't find any of them interesting to investigate further. A manually constructed input that is the same with input in question 23 reveals the vulnerability; this was manually found by GrammaTech.We have attached our KelinciWCA scripts.”

Evaluation:

Confirmed GrammaTech’s exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.2.2 Question 043 (AC Time, Intended Not Vulnerable, Answered: No)
Response:

"KelinciWCA generated several inputs, but we didn't find any of them interesting to investigate further. We wrote a driver to test the method processClause() in Arithmetizer class. We tested with the roman calculator and basic calculator. North easten mentioned that they discovered the input: 3^10413 * 115 *(2/2)^10709.The team used their tool haflfuzz to find thisinput. Justification could be that the division operator takes long. It contains calls to recursive functions and while loop. However, this exploit couldn't be confirmed on NUC."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.2.3 Question 045 (AC Space, Intended Vulnerable, Answered: No)
Response:

"KelinciWCA generated several inputs, but we didn't find any of them interesting to investigate further. We wrote a driver to test the method processClause() in Arithmetizer class. We tested with the roman calculator and basic calculator."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.5.2.3 *CyberWallet 1*

A.7.1.5.2.3.1 Question 006 (AC Space, Intended Not Vulnerable, Answered: No)
Response:

"Any file cannot be bigger than 10MB, like checked in this method: com.cyberpointllc.stac.webserver.manager.CompositeAssistant.HttpExchangeAppealContext.HttpExchangeAppealContext(HttpExchange)But the file is not part of the post request directly, it is just a temporary generated file. I do not understand enough about this mechanism to be able to retrieve any possible attack point, but it looks suspicous me.The method LoginManager.handleFetch also looked very suspicious because of a BigInteger calculation that might get extremely large in the method KeyExchangeServer.generateControlSecret, but was not useful because the call to WebServer has null passed as argument allowKeyFilethat prevents the method from being called."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.3.2 Question 035 (SC Space, Intended Vulnerable, Answered: Yes)
Response:

“When the client asks to display the account activity, the tier category is displayed in the feedback given from the server to the client and in theory that is a string that has different size but the activity also has all the accounts so the size can not be measured.

In collaborative engagement, Iowa and Grammatech showed that the image that is sent to the user is related to its tier. Low tier users get imageswith id 00,01,02,03 and highest tier users get 30-33 and the tier defined as a number between 0-3 defines the first digit of the image id. They also showed that each file had a distinct size and file size difference between tiers were at minimum 200K, so any small noise wouldn't affect it. We agree with these findings and believe that there is a side channel.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.3.3 Question 051 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“By manually inspection we found out that an attacker can transfer from any account, ifthe attacker knows the account_idof this other account. The account_id is a hashed value is nothing what a customer would actually see. But it is used in the messages to identify the account_id. In order to think about an attack, we tried to transfer money from an account_id, which does not exist, i.e. the

attacker tries to guess the account_id by checking the response time. The server checks the account_id by retrieving the Archive object for the corresponding account_id. This happens by calling the get-method of a HashMap. We analyzed the implementation of the hash map, but we did not find any sign for a timing side-channel there (see fuzzing results in artifacts folder: artifacts/q051/hashtable-fuzz-results-15min.zip).Another opportunity would be to guess the username and the password of another user, but the password comparison implementation seems to be safe (see fuzzing results in artifacts folder: artifacts/q051/passwordEqual-fuzz-result-15min.zip).Note: if you want to rerun the fuzzer, the instructions are documented in the folder, but be aware that you have to adjust the scripts.In collaborations, other teams did not find a side channel as well.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.4 *CyberWallet 2*

A.7.1.5.2.4.1 Question 024 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

“In our collaboration, Iowa team argued that the seed of the program is related to second digit of the image. (seeds arebetween 4-9) There is a relation between the file name and tiers with some collisions. One tier had a 3 way collision where we had to choose between 3 options and the least

occurring option occurred for 10.5%-11%, so it’s hard to guess it within the probability of success of 95%. Also, file sizes are randomized and there are some collisions there as well. Even

if there was a one to one relation between file size and file name, it would still be No. We could have run Profit to sniff the image sizes from the network and check if there is any correlation between the image sizes and the tier itself."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.4.2 Question 059 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"In the take home analysis, we didn't have time to look into this problem in more detail to run analyses.In collaboration, Grammatech team proposed that they found a side channel vulnerability. To transfer the money from someone else's account to your own account, either you need to guess username/password or set the amount of money transferred to another account negative and send negative amount of money to some other account number.Their findings were more on the second point where the money can be negative. When you call takeQuantity(), the quantity is multiplied with authorization multiplier. In BankerHelper class, there is an authorization field and if the authorization is successful, the multiplier is set to -1, otherwise, by default it is 1. Authorization number (in BankerHelper) is randomly set between 1-100, you can guess it by trial and error. After that, we need the account number.There's a side channel in time on account numbers where AccountConductor.findAccountFace() is the equals method for an account number and it has the segmented oracle vulnerability because in each iteration, the loop does a costly operation. Because the account numbers are 10 length, with digits between 0-9, it takes 100 operations from the budget. findAccountFace is calling testForDuplicateError and it always returns false after one loop that has a contains operation. They have also made an exploit that's working.To detect the side channel on equals method, we could have used CoCoChanel to find the path costs and noticed that the difference between paths is significant."

Evaluation:

Confirmed GrammaTech's explolit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.5 *Door Master*

A.7.1.5.2.5.1 Question 029 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"There's a suspicious loop in the KeyFobController.authKeyFobmethod; it loops endlessly as long as the result from the KeyFobService method is null; KeyFobService can return null if the KeyFobService.conn is false, which is the case if MessageLoop.pingfails (that is, the connection is terminated) or if the private key is not found in KeyFobService.keyidMap, which is performed when a key is deleted. So 2 potential attacks would be:1.add the key, start authentication and then quickly terminate the connection OR2.add the key and then send the commands to both start authentication and to delete the key simultaneously, so that there is a 50% chance that the key is deleted prior to the authentication completing Both cases were tested and we verified that the server is still able handle new requests because of multi-threading. Therefore no vulnerabilities were found."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.5.2 Question 041 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

“During the collaboration, two teams (Colorado and Northeastern) identified a timing side-channel through which one bit of the 4096-bit key can be recovered per-query through a segmented oracle attack. The suspicious part of the code was MessageLoop.run() where it sends pings with 1 second delay afterwards if no key is in KeyFobService and if there is a key in KeyFobService, then it sends pings with delays according to the KeyFobService.nextInt(). nextInt() function keeps track of the latest bit and returns 100 if the next bit is 0, 200 if the next bit is 1 where it is interpreted by miliseconds by the sleep function after pings. Using this, you can capture each bit of the key and when the bit sequence loops after a certain point, we can stop listening and reconstruct the key using the captured bits. Also some number of first bits of the key are same, so we can use that help reconstruction as well. Colorado also wrote an exploit for it where you can capture the bits, convert to ascii and save it in base64. The actual length of the key they were observing was around 2760-2768 bits (Not 4096 bits as there was a confusion with 512 bit exponent of the RSA key). We believe the key can be leaked.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.6 *EMU 6502*

A.7.1.5.2.6.1 Question 004 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“How else could I get to admin page?

1. Get the admin's password and log in as them.
2. 2. Get the admin's session (cookie)

Only weird part is the phrasing of the oracle question: "Query the oracle with a candidate username and receive a binary response (yes/no) as to whether it is the username of an admin user." Even if we query the oracle and get a positive response, that doesn't mean we have admin powers. There must be something tied to that. There doesn't seem to be any method of gaining access to either the admin's password or session, therefore there doesn't appear to have a vulnerability in this question.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.6.2 Question 012 (SC Space, Unintended Vulnerable, Answered: Yes)

Response:

“This program has a set amount of source files (~500) that gets loaded into the DB when first started. It doesn't look like any user, adminor not, can add any source files. Looks like the sizes of the files are in the range of [~7000,~16000] bytes, with only a few duplicate sizes (maybe 2 or 3 max). Once user selects a program,

they are taken to screen with 15-20 instructions loaded up. Can specify initial register values. User hits execute, instructions are executed one at a time. Sliding window moves with program counter (displays the current instruction +-10 instructions); as window moves, instructions are fetched from the server (in other words, user only sees 15-20 instructions at a time)●Instructions take ~1 second to execute, due to a thread.sleep(). Can be recompiled such that the sleep is gone. Then programs can be will executed blazingly fast. Profit can be used to profile instructions. Problem states that users can specify initial values. This is not the case: see Q014 for specifics. This means that each program is deterministic, and thus has a space signature. Multiple programs could have the same signature (collision)●After execute button is hit, server sends a push notification back with updates on the register values. Because we are constrained to interface with the program in the same manner as the example script which uses curl commands, EACH PUSH IS A DIFFERENT TCP CONNECTION.This means we need to separate program signatures into equivalence classes based on the subsequent register update. i.e., one program could start with updating registers 3, then 2, then 1 register, then 3 registers. Another could be: 2 registers, then 3, then 1, then 2, etc... We only have 25 operations, so its a balance between number of programs in equivalence class within subsequent push notifications (since each tpc connection is one operation).Northeastern ran an exhaustive test for each program (with the code patched to reduce the 1 second delay between steps to 0.3 sec to speed up the testing) to verify a distinct signature for each program. The results are expected to confirm that each program has a unique signature in the first 25 instructions such that the Oracle queries can be used to identify which program was run by the user under attack.”

Evaluation:

Correct Justification after review

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.6.3 Question 014 (SC Time, Intended Vulnerable, Answered: No)

Response:

“Depending on initial register values, number of instructions executed may not always be same, and program could go into infinite loop. Each execution always starts from line 1 however so that may be helpful.

●There are 4-6 registers which the user can specify, with each of size 8 bits. However, THIS VALUE IS NOT ACTUALLY USED TO INITIALIZE THE REGISTERS. Not sure if this is a mistake, but this means no matter what initial values the user tries to specify, the initial values in the internal registers are 0. The Emulatorclass holds the actual registers, and since they are private, the only way to access them are through the setRegX/getRegXmethods. setRegXis never called by the EmulatorView**

●Each program will take a certain amount of time, no matter how many times it is run. Thus, we just need to profile all ~500 programs with Profit and see how many collisions. If there are more than ~20 collisions, then its a NO. Otherwise, a YES. HOWEVER: After execute buttonis hit, server sends a push notification back with updates on the register values. Because we are

constrained to interface with the program in the same manner as the example script which uses curl commands, EACH PUSH IS A DIFFERENT TCP CONNECTION. That means we can only detect up to 24ish pushes! We probably can't get enough timing information from this, since one program takes 1s per instruction."

Evaluation:

Correct Justification after review

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.6.4 Question 016 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"There is also no mechanism to allow a user to download a new program, short of placing one in the data directory, which is not allowable. Emu6502 is very well constrained in that not only can't new programs be loaded or existing ones be modified, but also the registers that are allowed to be set by the user are never sent to the Emulator, and are never used. The registers are always set to defaults (0) prior to executing a program, and there is no mechanism to pause execution once started -only terminating it by terminating the connection. The only effective way of delaying the benign request is by exhausting the number of threads in the program prior to his issuing the command. This application is built on Spring Frameworks, which sets the size of the thread pool it uses. There is no override to its configuration, so it uses the default value of 200. Each command request would start 2 threads, which means we would need to have 100 users running to achieve this. Since our budget size is 5000 bytes, that allows us 50 bytes per iteration. Each iteration would need to login, choose a file to load (one that takes at least 60 seconds to execute) and then run the execution. This would not be possible to achieve."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.7 *InAndOut 1*

A.7.1.5.2.7.1 Question 007 (AC Time, Unintended Vulnerable, Answered: Yes)

Response:

"We looked for locations related to the pickup order functionality and noticed that the InAndOutAid.pickupOrder() method synchronizes on the binaryCodesInUse variable, thus meaning it blocks all other requests until finished. We also noticed that this method contains a loop that sleeps until an order is ready. Manual inspection shows that the binary code (computed from the order confirmation string) guards entry into this loop, but that the order confirmation is used to check whether an order is ready. Examining the getBinaryCodeFromValidationCode() method shows that there is a possibility for multiple confirmation strings to map to the same binary code; in particular, there appear to be 2^6 different binary codes possible, and it is very simple to generate all of these combinations (a "0" in the confirmation string represents a 0 in the binary expansion, otherwise it is a 1). An attacker can thus try these combinations and block the benign pickup request indefinitely since the binary code will pass the first check above but the order will never be ready since either (1) the processedOrders won't contain the order

confirmation number (likely), or (2) the attacker will guess the benign order confirmation number and actually take the benign user's pizza."

Evaluation:

Confirmed potential unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.7.2 Question 027 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

"Colorado identified a location in the code (CookingController.calculateStageSeries()) that adds to a Vector in a loop. This loop can be controlled by the choice of pizza toppings. They then manually tried all pizza topping combinations and found a particular combination of toppings that drives this loop high. Ordering 7 pizzas with those toppings, which is within budget, triggers the memory vulnerability. Our Janalyzer tool also identified this as a suspicious location in which an allocation is happening in a loop (see attached screenshots), but we did not investigate it in enough detail to identify the vulnerability initially due to the high number of such nested allocations (i.e., there were several false positives). Extending our tools to detect when looping allocations are in synchronized or locked blocks might help focus the analyst's attention on the correct spots initially."

Evaluation:

Confirmed Colorado's exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.7.3 Question 049 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"Also, it seems like if a user has someone else's order number, they can pick up the pizza, even though it's not the right username (as long as the order number matches, the pizza will be delivered).Note, however, that the question says we need to steal a victim's pizza and the victim is "identified by IP address". What does this mean? Aren't all clients supposed to be run on clientNUC?

During the collaborative engagement, we did a detailed analysis on this problem where we tried 100 random pizza order 6 times and quantified the information leakage where we removed the first 30 runs because JIT might affect it.

This is the leakage plot describing p(f|s) where f are the observables on the X axis for the feature "Timing delta between packets 1,2 in full trace, both directions", Y axis is the p(f|s) probability distribution, different colors show different secrets, and the points below show how the actual data points are distributed with some vertical fuzzing for distinguishing different points.

This shows that this program leaks. Although there are minor collisions, among all 570 points, they are at most 1% of the all data points and therefore we can say we can leak the tickets with timing side channels."

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.7.4 Question 055 (SC Space, Intended Not Vulnerable, Answered: No)
Response:

“The sizes of packets 1, 2, and 4 are constant. But the size of the 3rd packet varies. It seems to fluctuate between 90 and 120 approx. Also, it seems like if a user has someone else's order number, they can pick up the pizza, even though it's not the right username (as long as the order number matches, the pizza will be delivered).Note, however, that the question says we need to steal a victim's pizza and the victim is "identified by IP address". What does this mean? Aren't all clients supposed to be run on clientNUC? […] We can see from the results that this doesn't leak any information, at least not from the communication with the inventory manager.

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.8 *InAndOut 2*

A.7.1.5.2.8.1 Question 028 (SC Time/Space, Intended Vulnerable, Answered: No)
Response:

“We started by looking at the same two side channels that we had found in variant 1 of this application (seequestions Q049 and Q055), that is:

●the time elapsed between 2nd and 3rd packet exchanged between Server and InventoryManager

●and the size of the 3rd packet in said exchange

The time side channel, which was very reliable in Q049 (variant 1 of app), is completely noisy in variant 2. Since we have no oracle queries, it looks like this side channel has become useless. (Of course, there could be otherside channels in time. We should run Profit on this to make sure.)As for the side channel in space, in this variant it becomes MUCH stronger than in the other variant of inandout... But it is a bit noisy. With zero oracle queries, it does not look like this could reveal the secret with 70% probability of success. However, it narrows down the set of candidates a lot. So, if this were combined with an additional side channel (e.g., an additional timing side channel besides the one from Q049), it could enable an attack.

We also did one final analysis which also captured the communications between client and server. The leakage results are very similar, there is no new information.”

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.5.2.8.2 Question 039 (AC Time, Intended Vulnerable, Answered: Yes)
Response:

“We looked for locations related to the pickup order functionality and noticed that the InAndOutAid.pickupOrder() method synchronizes on the binaryCodesInUse variable, thus meaning it blocks all other requests until finished. We also noticed that this method contains a

loop that sleeps until an order is ready. Manual inspection shows that the binary code (computed from the order confirmation string) guards entry into this loop, but that the order confirmation is used to check whether an order is ready. Examining the getBinaryCodeFromValidationCode() method shows that there is a possibility for multiple confirmation strings to map to the same binary code; in particular, there appear to be 2^6 different binary codes possible, and it is very simple to generate all of these combinations (a "0" in the confirmation string represents a 0 in the binary expansion, otherwise it is a 1). An attacker can thus try these combinations and block the benign pickup request indefinitely since the binary code will pass the first check above but the order will never be ready since either (1) the processedOrders won't contain the order confirmation number (likely), or (2) the attacker will guess the benign order confirmation number and actually take the benign user's pizza."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.8.3 Question 053 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"Janalyzer revealed several looping allocations (see screenshot below). We investigated these allocation sites, but they did not lead to any vulnerability we could identify. "

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.9 *Litmedia 1*

A.7.1.5.2.9.1 Question 009 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"Application has a database that is growing with each bookmark (see data directory); bookmarks can alsobe deleted; the code is complicated so it could be that the deletion may fail but it is unlikely the

budget is exceeded. I looked a bit at classes BookmarkGuide and PublishingAtheneumActual. the application runs many threads in parallel. From the collaboration, the method PublishingViewerGuide.handleGrabadds image data to StringBuilder parameter sbfor each getRequestmade. Because the image files are rather large, making > 60 requests very rapidly can cause the memory to exceed 600 MB before the Garbage Collector has time to clean up.

Yes, by repeatedly issuing GET requests to the server very rapidly, you can exceed the memory budget before the Garbage Collector has time to free it."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.9.2 Question 032 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“We have written an app which simulates the interaction of the user logging in, checking the homepage, checking a specific literature andlogging out. We have created an input for each preference option in the media directory where it's the only preference option and its frequency is a

high number like 55. We have also created 250 inputs where there are 5 different preferences and its frequency values only differ by 1.We have run this on the NUCs using Profit where we ran each input 5 times for each of the 2 users.

It seems the most leaking feature is leaking 1.069 bits out of 5.907 bits. This means we have 4.838 bits of remaining unknown information. We can use 28-29 oracle queries to resolve the unknown information but our operational budget is 100 where1-10 operation is spent sniffing the network packets and we have 90-99 operations left (where each oracle query costs 10 operations). Therefore, with 9 oracle queries, we can't resolve this information unless there is some active operations where we can observe side channels and we don't know of any active operation which leaks this information.

In collaborative engagement, Northwestern said yes because they thought preferences and categories were one and the same but preferences are a tuple of category and list of tags. Bubblesort function seemed suspicious to them as well. After the user accesses a media, preferences get updated and a bubblesort function is called to resort the preferences. This would have been related to the secret but Grammatech mentioned the function only uses the frequency, so it's not dependent on the secret preference other than preference. Iowa also said no, the code was complex so they checked the timing and they couldn’t find any difference, supporting our findings.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.9.3 Question 056 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

“Number of possible secrets is 2^X where X is the number of media files (56), so it is impossible to cover all secrets in enough time. We have tried 300 sets of size 5 for bookmarks where each subset contains 5 media and populated the bookmarks of users and tested and profiled it using Profit where we ran each input 5 times for 2 different users (10 repeat runs per input in total).

Profit concluded that top 4 space features leaked 2.403 bits out of total information of 8.229 bits. (This is not equal to the real total information because we can only reason about the secrets we have seen.) We have 5.826 bits of information remaining which amounts to 57 guesses.Our operational budget is 200 operations and sniffing takes 1 operation and we can use 199 operations to use oracle queries. We can use all oracle queries to determine the secret. While it may need a more detailed worst case analysis, the application seemsto be leaking.

During collaborative engagement, everyone except us said yes, and showed that unlike litmedia_2, this version's bookmark page can contain up to 65 bookmarked items, so we would get a lot of collisions. Northeastern did the bucket calculations for bookmark pages with size 40

and found out if you calculate the page sizes, that there would be a 200 way collision with some of the pages. We did the Profit analysis with 400 bookmark pages of 40 elements and ran it 10 times (without ads as we can distinguish them) and got around 3 bits of leakage which would still make it within budget. To drive leakage down and cause a 200 way collision, we would need to generate a ton of inputs and where this experiment took 16 hours to run and 1 hour to analyze with because of high number of pcaps and large file size, running 1000+ elements is infeasible, so we concluded with the findings the other teams and changed our answer to No."

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.10 *Litmedia 2*

A.7.1.5.2.10.1 Question 001 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"We have written an app which simulates the interaction of the user logging in, checking the homepage, checking a specific literature and logging out. We have created an input for each preference option in the media directory where it's the only preference option and its frequency is a

high number like 55. We have also created 250 inputs where there are 5 different preferences and its frequency values only differ by 1.We have run this on the NUCs using Profit where we ran each input 5 times for each of the 2 users. […]

5 bits of unknown information, need 32 guesses to fully know the secret, only 9 oracle queries, can't handle it. In collaborative engagement, same with Q032, everyone agreed that the preference wasn't leaking."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.10.2 Question 050 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"We have created an input for each preference option in the media directory where it's the only preference option and its frequency is a

high number like 55. We have also created 250 inputs where there are 5 different preferences and its frequency values only differ by 1.We have run this on the NUCs using Profit where we ran each input 5 times for each of the 2 users. bits of unknown information, need 32 guesses to fully know the secret, only 9 oracle queries, can't handle it. In collaborative engagement, same with Q032, everyone agreed that the preference wasn't leaking."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.10.3 Question 058 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

“Because it says in the description that they check *their* bookmark list, we haven't explored incognito mode's effects. Number of possible secrets is 2^X where X is the number of media files (56), so it is impossible to cover all secrets in enough time. We have tried 300 sets of size 5 for bookmarks where each subset contains 5 media and populated the bookmarks of users and tested and profiled it using Profit where we ran each input 5 times for 2 different users (10 repeat runs per input in total). […]

After the profiling and analysis, Profit concluded that top 4 space features leaked 2.440 bits out of total information of 8.229 bits. (This is not equal to the real total information because we can only reason about the secrets we have seen.) We have 5.789 bits of information remaining which amounts to 56-57 guesses. […]

Our operational budget is 200 operations and sniffing takes 1 operation and we can use 199 operations to use oracle queries. We can use all oracle queries to determine the secret. While this question may need a more detailed worst case analysis, the application seems to be leaking. Post Collaboration update: Other teams (Northeastern, Iowa) have found out that there is a limit on number of bookmarks and it's 7. They agree that this question is a yes. Northwestern calculated the buckets of pages with the same size (without ads) and it's below 200. They used a sample of pages with 6 and 7 elements. Iowa did space analysis with a sample of pages with 2 elements. […]

On last day, Colorado pointed out that the C(65,7) is around 696 million and if we would have 200 elements per bucket, we would need 3.5 million buckets. We don’t think the there are 3.5 million pages with unique sizes. Both Northwestern’s, Iowa’s and our analysis depends on subsampling the actual input set and that made us overestimate the information leakage.”

Evaluation:

Correct through collaboration

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.11 *Phonemaster*

A.7.1.5.2.11.1 Question 003 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

“Applied DifFuzz on CheckLogin method in class Server (package com.ainfosec.phonemaster). Shows time for checking each user increases according to the position of the user in the user list. Timing information can therefore be used to guess the position in the list. The user list can be printed bythe attacker so the attacker can guess the user name of the target user.

In collaborative engagement, the others have also agreed with the findings. Colorado pointed out that if new users are added, the timing signature might change but we and Northeastern believed the timing signature is monotonic and regular with some little noise, so we could still predict the number of the new user and then get the list. For example, if the number of users increased from 10 to 50, the timing for user 5 would still takea path with cost proportional to 5 and a new user 25 would have a path with cost proportional to 25. Also we can do offline profiling with adding the users and profiling those users as well if we have the binary”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.12 *Powerstate*

A.7.1.5.2.12.1 Question 031 (AC Space, Intended Not Vulnerable, Answered: Yes)

Response:

“The application is rather small but contains many loops with complex logic (Janalyzer also reveals many loops but doesn't point out anything particularly suspicious for the majority of them). Similar to Q037, we conjectured that if we couldget the main PowerStateSolver.solve() loop to run many times, then it might trigger both a space and complexity vulnerability. We were not able to manually trigger such a vulnerability. However, we feel this program is a good use case for our new DSE tool and are trying to apply it.

During the engagement, Northeastern had a working exploit with a large number of buses. They found it by looking at loops and noticing something suspicious; they then manually crafted an input file. GrammaTech also manually found a much simpler exploit that just modified a few angles in original example input file.”

Evaluation:

Confirmed exploit for potential unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.12.2 Question 037 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

“So 4 long running requests should block a 5th (benign) request since the NUC has two cores plus hyper-threading (for a total of four available cores). (It appears very tough to get 5 simultaneous requests to increment the instanceCount to 5 before one of them can check the while condition above --the probability of success is 99%).This knowledge helps us limit the number of attacker requests to 4, leaving about 12.5kB per request.

if a certain condition is satisfied, the double-nested loop becomes a triple-nested loop. We believed a large input file that (1) sets nbus to a large value and (2) satisfies conditions like those above can drive the execution time high enough to exceed the budget.This program recompiles, so given enough time, confirming this suspicion is possible through (1) experimentation and (2) using our DSE tool. We tried applying the DSE tool to it, but did not have time to successfully generate an exploit.

During the engagement, Northeastern had a working exploit with a large number of buses. They found it by looking at loops and noticing something suspicious; they then manually crafted an input file. GrammaTech also manually found a much simpler exploit that just modified a few angles in original example input file; theycrafted this exploit using domain knowledge. Using this input four times (as we originally conjectured) does indeed block the benign request.”

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.13 *PSA*

A.7.1.5.2.13.1 Question 010 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"It seems that to come up with the exploit for this question, we need to play around with the thread pool and sending several requests fromclient to server. The key here is that we need to send several queries for the psa to analyze a module that is not in its database. The exploit is to do one POST request followed by 200 GET requests. By this way, the thread pool will be messed up and it will hang forever. This was a collaborative group effort done using manual inspection and testing."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.13.2 Question 030 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"An input that is within the budget and reveals the vulnerability is of the form of several nested '[]'. This input is manually crafted. There is a KelinciWCA-generated input that is a bit over the budget (12Kb) and reveal the vulnerability too"

Evaluation:

Confirmed this vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.14 *Roulette 1*

A.7.1.5.2.14.1 Question 013 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"See discussion for Q011. The side channel in that variant is no longer present. In the case of this variant of roulette, the spinTime which determines the winning number is given by a randomly drawn double. The generation of this double seems secure."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.14.2 Question 019 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"The only data written out as a file are composed of the ParticipantsDB which generates the following data per bid message issued (from ParticipantEncoder.serialize:

●a Long (has limited length, so we'll ignore this)

●a String composed of Participant.participantState (an enumerated value that also has limited length, so ignore)

●a String composed of Participant.publicEmpty

Participant.publicEmpty is setup from SenderReceiversEmpty, which consists of some fixed chars plus the String value of faceand code. faceis limited to a max length of 25, and although codeis composed of 2 BigIntegervalues that can be very large, these are the pand qvalues of the public key that are issued in each message, and therefore must be sent by the user. Increasing the size of these (if that is allowable) would not benefit since the budget size is 10 kB and we need to generate a file size of 1 GB or 100,000x this size."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.14.3 Question 048 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"See Q033. That version has a weak side channel that is not enough. This version does not even have that side channel. You can learn if a user won. This could potentially be used to get the type and value but not the amount. There are too many options for the amount."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.14.4 Question 054 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"If the attacker is allowed to modify his program version in order to send these byte sequence which triggers a long running procedure, then the request of the benign user will be blocked for the runtime. Unfortunately, we were not able to trigger a worst-case behavior just by manual construction. Nevertheless, the interesting part happens in the constructor of the RouletteMessage class.

Fuzzing: we wrote a (realistic=no modifcation in the target) driver, which mutates a byte sequence and creates a RouletteMessage with that sequence. It tries to maximize the instrumented cost. For a short runtime, the fuzzer can increase the runtime step-by-step, but it cannot create an attack sequence. But, the analysis gives a strong indication that this while loop can be used for the attack. […] There was not enoughtime to validate whether there is a specific sequence of commands that maximizes the execution time, so during the collaborative engagement, we pursued the idea of filling up the command queue such that additional commands (the benign request) would have to wait for

the other commands to be processed. The problem is that the queue depth of the wagers is 10000, so in order to fill up the queue would require 10000 wagers. With a maximum budget of 10 kB, it would not be possible to do this."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.15 *Roulette 2*

A.7.1.5.2.15.1 Question 011 (SC Space/Time, Intended Vulnerable, Answered: Yes)

Response:

“if you can time when the wagersOn reminder happens with some accuracy, you can predict the wheel spin!To prove that the perturbation didn't matter, we wrote a SPF Driver for the spin function where we sliced the spin function, set the perturbation variable to a symbolic real variable with range [-0.01, 0.01] and tested the spin function to see if there would be any change between spin results for different values of the perturbation. The results for our analysis show that any value of perturbation within that range would not change the result.

In collaboration, Grammatech showed that wagersOn reminder doesn't happen by default and needs to be activated by betting more than you have and pointed out that wagersOn variable is in seconds, so it can be measured pretty accurately.The secret seems to be leaking as you can use an active operation to activate the wagersOn reminder, time the wagersOn, and use the result of wagersOn to simulate the spin and know the result of the winning number beforehand.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.15.2 Question 033 (SC Space, Intended Vulnerable, Answered: No)

Response:

“messages like "*INFO: Your bet parity even 300 has been accepted your remaining bankroll is 970" message leaks some information. Packet size for "color red 4" is shorter than "color black 400". However, while this could be used to leak some info about the type of bet, there are too many possible bet amounts for this to be enough.

"color red XXX" is always the same length regardless of the value of the 3 digit bet. With basic analysis, this side channel doesn't seem to be enough as any of the 3 digit number combinations are confused with each other and there are 900 numbers from 100-999 and our budget is 255.We could have used Profit to analyze it but the basic analysis seems to be enough. Also, with the offline profiling, there is the manual cost of implementing the user interactions, defining variable input and secrets and writing mutators or input generators for it and there is a time cost of running the applications and doing numerical computations for information leakage calculation. In our collaborations, no other team have found a different side channel, so this remains the only side channel and it doesn't leak much information.”

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.5.2.15.3 Question 046 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

“The only data written out as a file are composed of the ParticipantsDB which generates the following data per bid message issued (from ParticipantEncoder.serialize:

●a Long (has limited length, so we'll ignore this)

●a String composed of Participant.participantState (an enumerated value that also has limited length, so ignore)

●a String composed of Participant.publicEmptyParticipant.publicEmpty is setup from SenderReceiversEmpty, which consists of some fixed chars plusthe String value of faceand code. faceis limited to a max length of 25, and although codeis composed of 2 BigIntegervalues that can be very large, these are the pand qvalues of the public key that are issued in each message, and therefore must be sent by the user. Increasing the size of these (if that is allowable) would not benefit since the budget size is 10 kB and we need to generate a file size of 1 GB or 100,000x this size.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.16 *Securgate*

A.7.1.5.2.16.1 Question 008 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

“In securgate there is an obvious space side channel that is due to the correlation between license plates and the size of the images of the plates. Ast the attacker has access to the images he/she can correlate the sum of the packets sent for each image (from server to client on GET_IMAGE request) with the actual plate number. Nico ran Profit over the example communication provided in the engagement and observed the following pattern (see his emails):

As for the total sum of bytes of all packets in each plate, yes indeed, it looks different for each plate!{4267192, 4266748, 4276092, 4258556, 4269276, 4254300, 4270860, 4270284, 4269276, 4206348, 4242716, 4271948}And that could be a signature. Additionally, there is the signature that I mentioned on my other email (the one with the plots) which is a more complex one—it has to do with the distribution of possible packet sizes within each plate.”

Evaluation:

Confirmed that this is the intended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.16.2 Question 018 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

“The code for get next plate is very simple; it does not seem plausible that there is an algorithmic complexity vulnerability. We did not examine this program in depth with tools due to a shortage of time. During the collaborative engagement, one team proposed an out-of-scope exploit in which the attacker would add a very large image to the directory of license plate images. However, this seems out of scope, and so the answer remains no.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.17 *Suspicion 1*

A.7.1.5.2.17.1 Question 020 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"This version of the program prevents the user from entering a negative repetition of a character, which allowed suspicion_2 to be exploited in Q060and Q044. No other exploits were able to be found, since Janalyzer was unable to run on this."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.17.2 Question 026 (SC Space, Intended Vulnerable, Answered: No)

Response:

"See Q057. The space side channel in that question has now been patched due to padding.In collaborative engagement, Iowa pointed out a possible side channel in distrustCompetition class where locationPointer variable keeps the secret and corresponds to the index of location in the fixed

list. sendAll() function sends the location to everyone, then the questions are sent when they happen. Iowa says the padding for the questions depend on base 3 of location pointer, in round X the length is increased with the value in X'th index of the location pointer. All the others including us didn'tfind any padding while examining it with Wireshark. Any question user can send, whatever the length, seemed to have a fixed size when we analyzed it when collaborating with Iowa. We did one final meeting and agreed on a No answer because we all couldn't find any side channel."

Evaluation:

Incorrect; they failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.5.2.17.3 Question 040 (SC Space/Time, Intended Vulnerable, Answered: No)

Response:

"See Q022.In collaborative engagement, Iowa showed their taint analysis results. Their taint analysis shows that password is used in DistrustCompetition.generateAndSendObjectives(), StartCompetitionCommand.execute, DistrustCompetition pullPassword, isPasswordCorrect methods. There are no loops dependent on password. Dynamic analysis we did showed (same in

Q022) that the size of password is leaking but that is not enough. Northwestern did analysis with tcpdump, didn't find anything. Grammatech did the taint analysis as well and got to the same conclusion with Iowa. We have an app but we didn't have time to run Profit with different passwords and check the leakage."

Evaluation:

Incorrect; they failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.5.2.18 *Suspicion 2*

A.7.1.5.2.18.1 Question 022 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"The password can be at most length 5. There is a side channel in space that leaks the length, but that is not enough. There does not seem to be any constraint on characters used in the password. The password checking function is secure. Dependency analysis shows no other place where the password is used.

In collaborative engagement, everyone agreed that there is very little side channel. Colorado was suspicious that there might be a crypto side channel related to fastModularMultiplyfunction for an adaptive attack, but on follow-up they concluded it would require more than 6000 operations, therefore, it would be out of budget. Same with Q040, we could have used Profit to test different passwords but we did not have time to run Profit on this problem."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.18.2 Question 044 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"There is a while loop in the decoding method transput.MorseCode.expand that outputs a value to a stream for a specified number of iterations that is specified by a value defined by the user. The number of iterations value is added to a parameter aggregateContent that is used to check for an

overrange amount of 1 million entries to prevent the user from simply entering a very large value, such as 2 billion. From visual inspection, it was observed that there is no check on the range of the iteration value itself, so specifying a negative number is allowable. This would then allow the user to specify an initial iteration count of negative 2 billion (which would cause no output to be generated) followed by an iteration count of positive 2 billion, which would then create a stream containing 2 billion characters, exceeding the memory limit. This would be allowable because the aggregateContent value would thing the total is 0."

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.18.3 Question 057 (Null SC Space, Intended Not Vulnerable, Answered: Yes)

Response:

"There is a minimum number of rounds that must be played before an accusation is made, meaning a certain number of answers must be sent. During collaborative engagement, Iowa team and we argued for yes and Iowa found the same padding we did. They did taint analysis focused on sendAll function and locationPointer. The only difference between this version and

suspicion_1 is the increasing of round counter. Whenever someone calls getPlays(), the number gets increased by 1 and returned as a result. If the users just ask their questions and accuse someone after 3 rounds like the rules of the game suggests, then the side channel is there.

There is one caveat, if the benign user accuses someone before 3 rounds are played, the code that checks that round counter is less than 3 uses the getter, increases the round counter and returns a warning to the user while not informing the other users. We can observe this from the network trace but the problem is that if a benign user is stubborn and they accuse someone 3 times in a row, then we lose the side channel. Kwame clarified that we should assume that the user behavior is described as the one in the scripts. We assume that the user doesn't act as stubborn as accusing someone for 3 times while getting warnings and there is a side channel if the users behave as expected and are abiding by the rules of the game."

Evaluation:

Potential unintended vuln not confirmed

Post-Engagement Analysis Ruling: Incorrect

A.7.1.5.2.18.4 Question 060 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"There is a while loop in the decoding method transput.MorseCode.expand that outputs a value to a stream for a specified number of iterations that is specified by a value defined by the user. The

number of iterations value is added to a parameter aggregateContentthat is used to check for an overrange amount of 1 million entries to prevent the user from simply entering a very large value, such as 2 billion. From visual inspection, it was observed that there is no check on the range of the iteration value itself, so specifying a negative number is allowable. This would then allow the user to specify an initial iteration count of negative 2 billion (which would cause no output to be generated) followed by an iteration count of positive 2 billion, which would then create a stream containing 2 billion characters, exceeding the memory limit. This would be allowable because the aggregateContent value would thing the total is 0."

Evaluation:

Confirmed potential unintended vuln

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.19 *ThermoMaster*

A.7.1.5.2.19.1 Question 021 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"Janalyzer reveals only one 2-level loop and 6 1-level loops. The log file is written to inside the run method, which is invoked on a new Thread each time the given PID is supposed to run. The server queues requests as they arrive (the Server.Response method is synchronized and the Thread.start()

is immediately followed by a Thread.join()), and the amount of data written to the log file looks fixed and bounded; for example, the example interaction scripts appear to write out the maximum amount of data to the log file and does not exceed the budget. Therefore it looks

unlikely that we can go over the limit. The collaborative group agreed that there appears to be no vulnerability."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.19.2 Question 038 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"We did dependency analysis using JAnalyzer to mark all secret dependent branches setting secret variable as this.setPointwhich leads to other variables such asthis.error, this.integral etc. that are dependent on the secret.

●Then, we used CoCo-Channel to detect side-channels (based on path imbalance analysis) setting cost threshold as 500 and number of loop iterations as 10 (parameters used before to detect side-channels for earlier engagement problems). CoCo-Channel reports that there are no side channels.

●We also recorded timing observations for different values of setPoint using the Wireshark tool. Based on timing differences we found that temperature values (in F) range [-32,-67] and [32,67] can be differentiated from other setPoint values. But this does not lead to any strong relation with setPoint by which we can discover another user's setPoint (target setPoint) to the nearest 1 degree Fahrenheit.

●Based on the results reported by the tools we used to analyze the system and observations about the session traffic, we conclude that there is no timing side-channel

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.19.3 Question 052 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Just submitting 5 relatively small attacker requests in this manner was enough to delay the benign request by around 10 seconds (the order in which queued requests were serviced was not deterministic). Therefore, we initially answered yes since more attacks would delay this even longer. During the collaboration, Northeastern said they tried the boundary cases for floating points. A request with a (positive) small floating point number for the current temp and a (negative) small floating point number for the set point runs for around 20 seconds, which is enough to delay the benign request by the limit. This appears to be caused by the cumulative execution time of (expensive) floating point computations. The group collaboratively looked for a reason that this input took a lot of time, and the conclusion was that the cumulative amount of expensive floating point computations pushed the time over the budget."

Evaluation:

Confirmed the exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.20 *Wordshare*

A.7.1.5.2.20.1 Question 002 (SC Space, Unintended Vulnerable, Answered: No)

Response:

“There are two kinds of search options: Search and SearchPriv. SearchPriv seems to enable a kind of segmented oracle like attack. If the secret is "abcde" and you search for "abc", it will be considered a hit. The current docs in the docs/ directory also seem to hint towards a segmented oracle attack, guessing one character at a time. The max length of the secret is 16 chars. However,

we discovered that SearchPriv can only be used by admin users, not just a user with the user key such as the attacker.To be able to use SearchPriv, the attacker must guess a secret key which hashes (under HMAC-SHA1) the search term prepended with some random string to the same value as the real secret key. We did not figure out how to do this.During the collaborative engagement, Colorado thought that if the attacker replaces the secret word with a Dictionary, then they could use a segmented oracle attack against the Dictionary. However, after group discussion and tests, we discovered that the differences they observe are related to the client not the server. The group could find no other side-channel.”

Evaluation:

Was originally correct but another team discovered an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.5.2.20.2 Question 005 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

“Janalyzer reveals a few recursion groups, but they appear bounded (e.g., it looks like there's no way to create a recursive cycle in the Trie).

Although an attack was found that succeeds in producing many files which may exceed the 35 MB usage limit, this does not comply with the question that requires that a single file is produced that exceeds this budget. The multiple files were generated by adding files with a name that contains many ':' chars, which outputs files based on the String.split for that character. The collaborative group could not identify a vulnerability.”

Evaluation:

Determined Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.20.3 Question 047 (SC Time, Unintended Non-Vulnerable, Answered: No)

Response:

“Due to time constraints, we didn't make a lot of progress on analyzing this program. It seems like you need the secret key in order to perform decryption of any secret word. However, the encryption seems secure to us. The previous side channel vulnerabilities in time are not present in this version.During the collaborative engagement, we discussed possible side channels. Iowa showed that WordShareDocControl.addfull() function decrypts the secret and adds to the Trie data structure but they didn’t find anything related to timing with this. Colorado found no timing difference between

different secrets. (Checked the time and space usage with secret.) For this question, there are two active operations you can do, either crack the password or private key or find timing difference with trying to add a secret word and it would have different timing if it's already in the tree but that doesn't seem to be case. When the search is done to fetch the words on a document, it goes over all the nodes over the tree and then filters out the secret words if the user is unprivileged. Grammatech's tool marked the tree class T9Trie suspicious, so they examined the side channel and claimed that the timing side channel is too small. Also if you can't bound the search, then any adaptive attack like lawDB is useless. There might bean attack with adding elements to T9Trie to overload the same paths (T9trie works like old phone texting where 1 is chars a,b,c, 2 is d,e,f, etc. builds chains from the root node to encode it with the phone style.) but we couldn't find anything, therefore the application doesn't seem to leak the secret."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.21 *YapMaster*

A.7.1.5.2.21.1 Question 015 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

"We originally looked at the MemoryMapWriter.put method, since that is where the yaps are persisted to disk; However, we were unable to find an exploit for this method.During the collaborative engagement, GrammaTech and Colorado said they manually noticed that the same memory-mapping is used when getting a yap and that a yap with a really high id (e.g., 900000) can be requested even if it doesn't exist. Reading a yap with this max id pushes memory usage to around 115MB, which exceeds the budget."

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.21.2 Question 017 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"Similar to 42: GrammaTech showed some suspicious behavior wherein they were able to modify the most recent yap, but they didn't have a way to read the most recent yap. No one could identify a side-channel observable through which the last yap could be read. Therefore we concluded no.

The collaborative group examined the traffic with Wireshark and observed that we are able to determine the user and id of the last yap, which means we need a key to decrypt it. While a timing side-channel to determine the key seems more likely to exist than a space side-channel (based on our observations), the group believe that 1250 operations would be enough to recover a key"

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.5.2.21.3 Question 042 (SC Space, Intended Vulnerable, Answered: No)

Response:

"There is space side channel from which it is possible to differentiate about the number of yap messages in inbox and based on that one can guess the id of last yap message, but we were not sure about space side channel for yap content or length of yap content.

During the collaborative engagement, GrammaTech showed some suspicious behavior wherein they were able to modify the most recent yap, but they didn't have a way to read the most recent yap. No one could identify a side-channel observable through which the last yap could be read. Therefore we concluded no. The group examined the traffic with Wireshark and observed that we are able to determine the user and id of the last yap, which means we need a key to decrypt it. We didn't see a space side-channel that could reveal the key, though."

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6 Two Six Labs

A.7.1.6.1 Answer Statistics

Table A-153: Engagement 7 Two Six Labs Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	11	9	81
SC Time	13	10	77
SC Space/ Time	7	5	71
AC in Space	18	14	78
AC in Time	11	8	72
Total	**60**	**46**	**77**

Table A-154: Engagement 7 Two Six Labs Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	30	16	53
Not Vulnerable	30	30	100
Yes Answer	17	16	94
No Answer	43	30	70

A.7.1.6.2 Two Six Labs Specific Responses

A.7.1.6.2.1 Calculator 3

A.7.1.6.2.1.1 Question 023 (AC Space, Unintended Vulnerable, Answered: No)

Response:

"No. We found no mechanisms in either the pre-processing logic or the calculator operation that would cause the AC memory effect. Of note the mechanism introduced in the pre-processing logic in calculator_4 that was used to solve Question_036 is not present in calculator_03. The greatest memory effect we were able to generate was in the 100Mb range."

Evaluation:

Was originally correct but another team discovered an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.1.2 Question 025 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

" No. The only file that is written to by the challenge program is a log file from the log4j library. This file is written to in the event of an InterruptedException or an ExecutionException from the call to Future.get() in the AbstractHttpGuide. InterruptedExceptions occur when a thread is waiting, sleeping or otherwise occupied and is then interrupted. There are no custom exceptions in the challenge program that extend InterruptedException or ExecutionException. We were able to trigger an ExecutionException by causing a stack overflow, but this utilized tens of thousands of bytes of the input to generate a modest effect (under 100k) on the file."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.1.3 Question 034 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Total PDU size 1697 B according to Wireshark, well under the 10 kB limit.

In the LargeNumeral class, the isEven method is incorrectly

implemented where it only considers numbers ending in 0,2,4 and 6 to be even (neglecting numbers that end in "8") as seen in the inequality governing q in the for loop below. When base=10, base/2-1=4 and the the inequality is not inclusive.

```
for (int q = 0; q < base / 2 - 1; ++q) {
   if (lastDigit == q * 2) {
      return true;
   }
}
```

This issue causes the root method to allow nth roots of negative numbers when n ends with 8. Even roots of negative numbers are imaginary, so it's no surprise that the implemented integer root algorithm has strange behavior in this case. The root method contains the following while loop:

```
while (!ONE.hasGreaterMagnitudeThan(delta)) {
   final LargeNumeral p = val.exp(n - 1);
   delta = this.divide(p).subtract(val);
   delta = delta.divide(new LargeNumeral(n, this.base));
   val = val.incorporate(delta);
   if (val.pullDegree() >= 5) {
      return this.rootByContinuedFractions(n);
   }
   if (val.isZero()) {
      return val;
   }
}
```

where n is the exponent, "this" is the given number we are taking an nth root of and val is 3 *(the number of digits of "this") / n. If this is significantly large (and negative) and n is small, the above while loop fails to escape. Experimentally we determined that the operation '(0-9999999)r8' more than exceeds the target runtime."

Evaluation:

Correct; found the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.2 Calculator 4

A.7.1.6.2.2.1 Question 036 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

"In the Arithmetizer class, in processClause, when the program encounters an operator, it looks ahead using getFollowingIntChunk. If you supply a valid computation that begins with many operators and contains a large numerical value (as a String), each call to getFollowingIntChunk causes a large amount of memory to be allocated. Our input to the calculator program consists of 3000 "(" followed by 3990 "9" followed by 3000 ")", which the calculator correctly evaluates, but uses over 4GB of memory to do so."

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.2.2 Question 043 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"there is an AC time effect, but we could not get the challenge program to utilize the desired amount of CPU time within the given input budget. The vulnerable code is contained in the Arithmetizer.processClause method. When presented with an input string that begins with a large number of operator characters (e.g. "+-()r^"), the while loop in processClause iterates for each operator. By appending a long integer to the end of the input string, the getFollowingIntChunk takes a large amount of CPU time for each iteration. Incidentally, this process also generates the AC space effect for Question_036. There is a CPU time trade-off between the length of the operator string at the beginning of the input and the length of the integer at the end of the input string. […] As we can see, there is a parabolic effect in tradeoff between operators and integers, but none of the given input eclipsed the 120 second target. Given an input budget of around 13k, we were able to achieve the target:

print ('+'*4000 + '9'*8000) 2k over input budget (at least) gets 167.46 seconds"

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.2.3 Question 045 (AC Space, Intended Vulnerable, Answered: No)

Response:

"The only file that is written to by the challenge program is a log file from the log4j library. This file is written to in the event of an InterruptedException or an ExecutionException from the call to Future.get() in the AbstractHttpGuide. InterruptedExceptions occur when a thread is waiting, sleeping or otherwise occupied and is then interrupted. There are no custom exceptions in the challenge program that extend InterruptedException or ExecutionException. We were able to trigger an ExecutionException by causing a stack overflow, but this utilized tens of thousands of bytes of the input to generate a modest effect (under 100k) on the file."

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.3 CyberWallet 1

A.7.1.6.2.3.1 Question 006 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"We investigated several interesting code paths, but were unable to find any AC space vulnerabilities. The transfer page allows for an unbounded field for the amount of money. We tried sending inputs that would result in very large or very small numbers, and neither caused a measurable effect. We also noticed that the server accepts malformed requests where the attacker can add a list of buttons which causes the challenge program to iterate over a large list of UI elements. None of these attempts had any meaningful effect on memory usage. In addition, the majority of the memory used by the challenge program comes from IO operations involving the images sent back with each request. We did not find any logic that would send multiple images, nor any mechanism for adding or augmenting images of the available pool. "

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.3.2 Question 035 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

“The size of the advertisement displayed when the users view their
account summary is dependant on the account balance. The important
calculations occur in ArchiveInfoManager.handleFetch(), which handles get
requests for the /accounts endpoint. The additional background tells us
the attacker will go here to check the target account balance. First, the
balance variable is set to the user's balance / 1000 dollars. Then, a
random value is chosen and added to balance. The random value, however,
is a Gaussian with mean 0 and standard deviation 1, so it is unlikely to
significantly change the calculation. We will return to an analysis of the
edge cases where it does change the calculation later on. In the next step,
picker is set to the log base 10 of the previously calculated value rand.
Picker is rounded down to the nearest integer, and if picker is greater
than 3, it is set to 3. This gives us the following relationship between a
user's balance and picker (again, dismissing the gaussian for now):

balance $[0, 10,000) --> rand < 10 --> picker <= 0
balance $[10,000, 100,000) --> 10 <= rand < 100 --> picker = 1
balance $[100,000, 1,000,000) --> 100 <= rand < 1000 --> picker = 2
balance > $1,000,000 --> rand >= 1000 --> picker = 3

Note that this corresponds perfectly with the account tiers.

The picker value is used to select which adds will be displayed to the user. if picker is 1, then takeEnsuingAd will randomly

choose between ad01, ad11, ad21, ad31, and ad41. Note that there is no overlap in the sizes among the sets, and the smallest gap

between any two sets is over 170000 bytes. This is much larger than any other
content on the account summary page. Thus, an attacker can use the total size
of the packets returned when a user views the account summary page to identify
the size of the chosen advertisement, use the table above to determine the

picker value, and use the picker value to discover the tier of the target user's account all using 1 passive operation. ”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.3.3 Question 051 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“The server calculates a secret authorization number when creating the BankerAssistant. BankerAssistant.authorization is a random value between 0 and 99. The value is used in BankerAssistant.isAuthorized, which is called from ExchangeCore when creating a money transfer. Though it is not present in the user interface, a user can submit a transfer request with an authorizationNumber field, and if the submitted value matches BankerAssistant.authorization, then this.authorized is set to -1 instead of the normal value of 1 in ExchangeCore. The grabQuantity() method returns this.authorized * this.quantity, so submitting the correct authorizationNumber causes the transfer amount to be negative, which is normally not allowed. The result is that a user who knows the authorizationNumber can submit a transfer of negative value to any target account, which has the effect of subtracting money from the target account, and adding money to the attacker's account.

Because there are only 100 possible values for BankerAssistant.authorization, and the attacker has 5000 operations, the attacker can brute force the value by submitting 100 transfers with authorizationNumber between 0 and 99. The attacker can review whether the amount of the transfer is positive or negative and then cancel the transfers before they are submitted to avoid sending money out of their account. Once the attacker has learned the correct authorizationNumber, all they can use the target account information to transfer money from the target account into their own. We have implemented this attack in exploit.py.

The attacker can transfer any amount up to their current account balance into their account, so they can double their account balance with each transfer.

Note that the server does not check the value of the target account, so it is possible to overdraw the target account, which will then have a negative balance."

Evaluation:

It seems that they misinterpreted the question to mean that the target account (account and routing numbers) were given. Given that assumption, no SC is needed.

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.4 CyberWallet 2

A.7.1.6.2.4.1 Question 024 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"There is no side channel which would allow an attacker to discover the tier of a target account. We know that the user will visit the /accounts endpoint to check the account balance of the target account. With only 2 operations, we determined that our only chance for a side channel would be in the handler for this request. We noted that the account details include both the account tier and the account balance, however there are multiple other fields with variable length including transfers completed today, cash withdrawn today, and the account creation date. In AccountDisclosureManager.handleGet(), these fields are all combined together in a single StringBuilder object, and sent in the same response packet. Furthermore, checking, savings, and CD accounts all have different text, and the many possible variations would mask any information about account balance or tier. There is also an ad loaded on the page. We looked at whether the size of the sponsored content leaked any information about the account. The account balance is used in the calculation of which ad to display, however worth is multiplied by Math.abs(bankerHelper.getEnsuingRandom()). BankerHelper.getEnsuingRandom() returns random.nextGaussian(), which is a random gaussian value with mean 0 and standard deviation 1. Multiplying by this value effectively masks the account balance. We conclude There is no side channel in space from which an attacker can discover the tier of a target account."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.4.2 Question 059 (SC Time, Intended Vulnerable, Answered: No)

Response:

"There is no side channel in time that would allow a user to discover information that would enable them to transfer money from another user's account into their own. We analyzed the code for logging in to the server and determined there is no leakage of information about a user's password that would enable an attacker to log in as the user. Since this is the only action the attacker is guaranteed to observe, we concluded that there is no side channel vulnerability in time in the application. We also looked for but did not find any other user actions in the application that could constitute a side channel in time when observed by an attacker. However, we did identify an attack that allows an attacker to transfer money from a target account into their own account, which we detail below. This attack does not rely on a side channel vulnerability in space or time. [… Continues to include the Non-SC vuln described in Q051]

Evaluation:

It seems that they misinterpreted the question to mean that the target account (account and routing numbers) were given. Given that assumption, no SC is needed.

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.5 Door Master

A.7.1.6.2.5.1 Question 029 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"The codebase is very small, and so it has been exhausted by manual analysis. Additionally, there are not many variables that can be controlled through user input, and we only get to send about 4 requests based on our input budget.

While MatrixRandomizer always returns 0 and is not random, unfortunately there is only place where this random is used - in the MessageLoop method to be an argument to thread.sleep. While this originally looked promising, it appears that thread.sleep always sleeps for a constant 100ms, and there is no way to influence the output of MatrixRandomizer.

Additionally, the KeyFobController class had a nested while loop that was of interest, though there is no mechanism to trigger that loop in a favorable

way.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.5.2 Question 041 (SC Time, Intended Vulnerable, Answered: No)

Response:

“It appears that through observing both the target user's initial setup request (mainly a call to KeyGen) and their key authorization request, there is no information leakage in timing. Nearly all critical operations are fast HashMap table lookups. As is the case for question 29, the hand-rolled PRNG is only used in one call within MessageLoop that always uses a constant value. Additionally, the codebase is small enough that we have fully exhausted it via manual analysis.

Of note - on every call to KeyGen, the hand-rolled PRNG seed is set to the previously generated private key. However, since key generation ultimately uses Java SecureRandom, there does not seem to be anything we can observe, especially using timing.”

Evaluation:

Incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.6 EMU 6502

A.7.1.6.2.6.1 Question 004 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“All user-authentication is performed using out-of-scope Spring classes (which did not appear to be vulnerable anyway). We found no mechanism to delete another use without logging in as an admin user. Cookie highjacking allows one to impersonate a signed in admin user, but there is no guarantee that an admin user is active, or that we could access the cookie.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.6.2 Question 012 (SC Space, Unintended Vulnerable, Answered: Yes)

Response:

"When the user selects a program and visits its page

> the emu6502 program sends a large (~1500-2000 byte) packet containing information to populate the program interface. Once the user clicks the `execute` button to begin emulation of the program emu6502 sends a new UI update packet every second thereafter (after emulating each instruction). These packets contain the information needed by the Vaadin client software running in the browser to (re)draw and update the UI. Importantly, these packets contain the 15 lines of program source code that are to be displayed to the user. Regardless of which lines were displayed at the previous timestep, each packet contains all 15 lines around the most recently emulated instruction. As the sizes of these 15 line instruction blocks are relatively varied and the instruction sequence of each program is deterministic, the lengths of the packets in the packet sequence for each program provide a unique fingerprint. […]

In the decrypted packet below, the program instructions can

> clearly be seen, starting with `DEY`. At the end of the packet are the `timing` values `[2746, 3]`. The combined length of these values varies from the 5 shown here (4 digit number + 1 digit number) to a maximum of 8 (4 digit number + 4 digit number). Other values that may change in length include the `syncId`, which appears to monotonically increase with each update packet and thus increases in length by one byte when going, eg, from `99` to `100`. Despite these variables, the size of the packet is dominated by the program instructions, enabling our side channel attack."

Evaluation:

Justification correct following review

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.6.3 Question 014 (SC Time, Intended Vulnerable, Answered: No)

Response:

"When a source is selected for emulation, the challenge program

enters a loop, which executes once per second until the challenge program finishes execution (i.e. once per instruction throughout program execution). This occurs in the run() method of CodeExecutionThread. The number of times this loop executes is a timing side channel which uniquely identifies the selected source. We verified this using a modified challenge program that removes the one second sleep statement for each iteration, as the emulated sources typically take tens of minutes to run. Using the modified challenge program we were able to determine that the length of the loop terminates either with the graceful termination of the emulated program, an index out of bounds exception (if the "memory" variable grows too large) or if the number of executions reaches the length of the challenge program (in lines). These values are distinct or have very small binning across the 500 selectable sources.

However, the above side channel cannot be viewed using the given input budget and with the clarification that the benign user is communicating with the server via curl requests. Using this definition of user interaction, the attacker must use a passive operation to observe each and every register update (each iteration of the above loop). If the benign user were interacting through a web browser, then the side-channel attack above could work (modulo tls cipher spec renegotiations).
With the restriction of the benign user using the curl scripts, one can only view the first ~25 register requests. These individual requests are almost constant time and do not have interesting timing variation."

Evaluation:

Correct following review; they may have misunderstood a 'clarification'?

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.6.4 Question 016 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"The emulation code is all constant time, and execution of the 500

given sources is deterministic. Since user's can't upload their own sources, the only attack vector involves interacting with the fields on the various user pages. We did not find any mechanism that would generate an AC time effect. Even if we did, past experience with Vaadin based applications indicates that an AC effect may not even necessarily block the benign request."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.7 InAndOut 1

A.7.1.6.2.7.1 Question 007 (AC Time, Unintended Vulnerable, Answered: Yes)

Response:

"Yes. There is an error in the verification of a validationCode which allows an attacker to permanently delay subsequent order requests. An attacker can order a pizza and then submit a pickup order using the resulting validation code. Instead of using the exact validation code, the attacker should change any non-zero digit in the validation code to a different digit. This will prevent any subsequent pickup requests from being processed. The attack works because of an imprecise check occuring in the InAndOutAid.pickupOrder() method when verifying the provided validation code. Validation codes are formed from the binary representation of a random number between 1 and 63. The code for an integer n is x_0-x_1-x_2-x_3-x_4-x_5, where x_i is either 0 or 2^i depending on whether the corresponding bit of the binary representation is 0 or 1. For example, 38 in binary is 100110, which gets transformed into 0-2-4-0-0-32. When checking a user's code, pickupOrder() only verifies that the string is the correct length and matches the original number when transformed back into binary. So for the example above with 38 as the chosen number, all of the following strings will pass the binaryCodesInUse.isCodeInUse() despite not being the actual code: 0-1-4-0-0-32, 0-3-4-0-0-32, 0-9-9-0-0-99, 0-1-1-0-0-11, etc.

After passing this check, however, the code enters an infinite loop, and returns only when the submitted validation code is present in the list of

processed orders. Unlike the previous check, presence in this list is detrermined only by an exact string match. So any of the altered values above will throw the program into an infinite loop. Although there is a one second sleep that could allow other requests to be processed, this loop is part of a critical section of code designated by the keyword `synchronized (this.binaryCodesInUse)`, so any actions requiring this resourse, which includes ordering and picking up pizza, will be blocked forever."

Evaluation:

Potential unintended vulnerability confirmed

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.7.2 Question 027 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

"Yes. In the CookingCONTROLLER calculateStageSeries() method, there is a do/while loop that continues to add an element to a Vector object. By causing "toggleConnector" to be large, an attacker can cause an AC memory blowup. The toggleConnector is set in OrderDirector.pullOvenCookingSwitch. The ovenCookingSiwtchOffset that is calculated there must be greater than zero for this effect to take place. By selecting a pizza with the toppings "Extra Cheese, Falafel, Pepperoni, Pineapple and Tomato Sauce", and ordering a quantity of two pizzas, the ovenCookingSwitchOffset is set to 8. This causes the returned value of OrderDirector.pullOvenCookingSwitch to be 90088140 (rather than the typical 263903). The numerical values of the return are determined by the hard-coded SETTINGS in CookingSettings.

example effect on nuc as read with pmap -x:

Address	Kbytes	RSS	Dirty	Mode	Mapping
total kB	16369752	3064628	3049772"		

Evaluation:

Exploit confirmed

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.7.3 Question 049 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes. The confirmation code generated after a user submits a pizza order to the InAndOut web server contains a distinguishable delay before the confirmation web page is sent to the user. […] After creating 1000 separate pizza orders to record the timing delays, we were able to create a chart of validation codes and corresponding times with small amounts of deviation. When sorted from the shortest to longest time delay, the validation codes were lined up numerically by the respective random value used to generate that validation code, from 1 to 63. With a budget of 10 operations, we were able to successfully submit a pickup request within 5 pickup attempts. We provide the script used to collect the timing delays of pizza orders.

The file "data.tsv" provides an example of what this data looks like. The attack would use 1 passive operation to obtain timing information, and the remaining 9 would be used for active operations to attempt a pick-up. We would traverse our data in order of the closest timing to what is seen from the passive operation. As mentioned previously, in practice we can do this within 5 attempts."

Evaluation:

Confirmed their exploit of mapping timings to the pickup codes

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.7.4 Question 055 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"No. After a user submits a pizza order to the InAndOut web server, the confirmation code to pick up the pizza is displayed in a new web page at the following URL https://serverNUC:8443/confirmation. This web page is generated using a template named orderconfirmationpage.html and is instantiated with several fields.

The data for each field remains the same with the exception of the confirmation code. Although we are not able to view the contents of the

confirmation page due to TLS encryption, we can observe the packet frame size of the response to determine the size of the confirmation code.

There are a total of 63 different confirmation codes that take the form of 1-2-4-8-16-32 where each digit is either the corresponding positional number or 0. The lengths of confirmation codes can be either 11, 12, or 13 characters long since the numbers '16' and '32' need two characters to be represented but may not be present. The number of confirmations codes with length 11 is 15 (omitting the invalid code 0-0-0-0-0-0), length 12 is 32, and length 13 is 16. Given the operational budget of 10, we are not able to reduce the space of possible validation codes for a target IP address any further and can not meet the 70% probability of success."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.8 InAndOut 2

A.7.1.6.2.8.1 Question 028 (SC Time/Space, Intended Vulnerable, Answered: No)

Response:

"No. After a user submits a pizza order to the InAndOut web server, the confirmation code is sent back to the user in a confirmation page. This web page is generated using an HTTP template instantiated with several fields that remain constant with the exception of the confirmation code. Due to how confirmation codes are represented, we can observe the network traffic for the confirmation page and determine the correct length of the confirmation code by the size of the packet frame. […] We attempt to reduce the valid confirmation code space further by analyzing the time delay between submitting a pizza order and receiving the confirmation page. After recording 1000 separate pizza orders, the results showed standard deviations in time as high as 300 milliseconds for a single confirmation code. The shortest and longest mean delay time are separated by only 380 milliseconds, signifying that timing for confirmation codes have tremendous overlap. Using manual code analysis, we found that the processing of orders are delayed by a poisson distribution calculated in the

pullPoissonRandomDeviate() method and may add an additional processing time delay of 0 to ~3200 milliseconds.

Due to the randomization of the pizza processing delays, we are not able to leverage the consistent timing of the calculateValidationCode() method used in Question_049. The poisson distribution and processing delay successfully obfuscate the timing of the calculation for a confirmation code, thus we conclude no side channel vulnerability exists in space and time for the given input budget and probability of success."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.8.2 Question 039 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes. There is an error in the verification of a validationCode which allows an attacker to permanently delay subsequent order requests. An attacker can order a pizza and then submit a pickup order using the resulting validation code. Instead of using the exact validation code, the attacker should change any non-zero digit in the validation code to a different digit. This will prevent any subsequent pickup requests from being processed. The attack works because of an imprecise check occuring in the InAndOutAid.pickupOrder() method when verifying the provided validation code. Validation codes are formed from the binary representation of a random number between 1 and 63. The code for an integer n is x_0-x_1-x_2-x_3-x_4-x_5, where x_i is either 0 or 2^i depending on whether the corresponding bit of the binary representation is 0 or 1. For example, 38 in binary is 100110, which gets transformed into 0-2-4-0-0-32. When checking a user's code, pickupOrder() only verifies that the string is the correct length and matches the original number when transformed back into binary. So for the example above with 38 as the chosen number, all of the following strings will pass the binaryCodesInUse.isCodeInUse() despite not being the actual code: 0-1-4-0-0-32, 0-3-4-0-0-32, 0-9-9-0-0-99, 0-1-1-0-0-11, etc.

After passing this check, however, the code enters an infinite loop, and returns only when the submitted validation code is present in the list of processed orders. Unlike the previous check, presence in this list is detrermined only by an exact string match. So any of the altered values above will throw the program into an infinite loop. Although there is a one second sleep that could allow other requests to be processed, this loop is part of a critical section of code designated by the keyword `synchronized (this.binaryCodesInUse)`, so any actions requiring this resourse, which includes ordering and picking up pizza, will be blocked forever."

Evaluation:

Confirmed exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.8.3 Question 053 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"No. Neither the InAndOut web server or the inventory manager were able to reach the resource usage limit of 2 GB. Once a pizza is ordered, that order is then added to a Queue data structure before it is processed and ready to be picked up. The maximum number of pizza orders that can be added to this Queue is bounded by the number of possible confirmations codes; 63. When this limit is reached, all confirmations codes are exhausted and the program is only able to reach roughly 280 MB of memory usage as all new pizza order requests are not saved. Using different variations of pizza toppings and quantities showed little to no effect on memory usage."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.9 Litmedia 1

A.7.1.6.2.9.1 Question 009 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

"Yes. We found a way to send multiple add bookmark requests in a single post. In PublishingViewerGuide.handlePost(), sending "add boomark" in a form field creates and invokes a new PublishingViewerGuideGateKeeper. The post handler

processes all fields named "add bookmark" in a while loop. The 10kB input budget is large enough to put 60 add bookmark requests in a single post, which is enough to cause the memory usage to grow to 600 MB."

Evaluation:

Confirmed exploit for potential unintended vulnerability.

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.9.2 Question 032 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"No. See explanation from question 50 - based on the same experiment and emperical data, we conclude that there is no side channel vulnerability in time.

While the question specifically asks for the top-ranked stored preference, which is subtly different than the top-ranked literature file, it is close enough that the experiment and data still applies. The reason for this is because a "preference" is simply a pair of a category, and a tuple of tags (the tuple of tags can be 1, 2, or 3 tags long), which almost always equates to one possible piece of literature in. There is, however, a very small amount of preferences with two pieces of literature in it.

The same experiment from question 50 was performed, and the data generated is included here as data.tsv. The data had to be re-generated since the previous experiments were for litmedia_2, whereas this is for litmedia_1.
All timing measurements are in milliseconds."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.9.3 Question 056 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"No. During login, homepage request, and bookmark page request, the same processing is performed regardless of the target users bookmark list. The only aspect of processing that could potentially have an effect time-wise is the loading of the bookmark page. However, retrieving the bookmark page is a fast

operation - it is simply getting a precomputed list. Emperical timing of the retrieval shows that it takes the same amount of time regardless of what is actually in the bookmark list. […] From the length data, we conclude that there is no side channel in space, based on the fact that we can construct a bookmark list that would be impossible to determine within the operational budget. From the data, we observe that there are 7 entries of size 16, 5 entries of size 18, 5 entries of size 20, and 6 entries of size 25. We note that bookmark lists can have up to 40 entries.

Given these lengths, there are 7*5*5*6 = 1050 possible bookmark lists of total size 79 (where 79 = 16+18+20+25). Thus, if we were to see a bookmark list of size 79 over the wire, we would have a very low probability of correctly guessing the entire bookmark list with 200 oracle queries."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.10 Litmedia 2

A.7.1.6.2.10.1 Question 001 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"No. See explanation from question 50 - based on the same experiment and emperical data, we conclude that there is no side channel vulnerability in time.

While the question specifically asks for the top-ranked stored preference, which is subtly different than the top-ranked literature file, it is close enough that the experiment and data still applies. The reason for this is because a "preference" is simply a pair of a category, and a tuple of tags (the tuple of tags can be 1, 2, or 3 tags long), which almost always equates to one possible piece of literature in. There is, however, a very small amount of preferences with two pieces of literature in it.

The same data from the experiment in question 50 is included here as data.tsv. All timing measurements are in milliseconds."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.10.2 Question 050 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"No. By manual inspecting and exhausting the code path taken of the target user signing in, we concluded that the sign-in did not leak any information via timing.

We also concluded that the target viewing a piece of literature does not leak any information via timing. This was confirmed by conducted the following experiment - using Byteman, we timed the execution of obtainMediaFile(). By our analysis, this was the method where nearly all of the work is done in when a user requests a piece of literature. We used a bash script to simulate a user logging in and then viewing a piece of literature, and had this done for all 65 pieces of literature, 8 times for each piece of literature. The timing was obtained, for each of the 65 pieces of literature we used the 8 timings to get a rough estimate of mean and standard deviation. It should be noted that between each run, we reset the database used by the program so that user "preferences" are not updating on each run and influencing timing. […]"

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.10.3 Question 058 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"No. During login, homepage request, and bookmark page request, the same processing is performed regardless of the target users bookmark list. The only aspect of processing that could potentially have an effect time-wise is the loading of the bookmark page. However, retrieving the bookmark page is a fast operation - it is simply getting a precomputed list. Emperical timing of the retrieval shows that it takes the same amount of time regardless of what is actually in the bookmark list. […] Given these lengths, there are 4*8*5*4*6 = 3840 possible bookmark lists of

total size 136 (where 136 = 22+25+26+28+35). Thus, if we were to see a bookmark list of size 136 over the wire, we would have a very low probability of correctly guessing the entire bookmark list with 200 oracle queries."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.11 Phonemaster

A.7.1.6.2.11.1 Question 003 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes, there is a side channel vulnerability in time from which an attacker can discover the username of a target user submitting any one of the requests demonstrated in simulateUsers.sh.

Each request demonstrated by the simulateUsers.sh script begins with the username and password of the target user. The `Server::serve()` method passes the provided username and password to `Server::CheckLogin()`, which loops over each user known to the Phonemaster program and checks if the provided password is valid for that user by calling `Server::checkPassword()`. `checkPassword()` calls `Server::hashPassword()` on the provided password then compares it to the stored value for the given user. `hashPassword()`, as its name implies, hashes the provided password with 50000 rounds of MD5. Note, though, that before calling `checkPassword()`, `checkLogin()` calls sleeps for `PAD_MILL` milliseconds. `PAD_MILL` is set at program startup to a random value between `0` and `1001` via a call to `ThreadLocalRandom.current().nextInt()`. While random, this value is constant for the lifetime of the Phonemaster program, as it is not changed after program startup. […] To rephrase in more purely mathematical terms, the login time for a user `u` is `L_u = PAD_MILL + p_u * HASH_TIME`. `p_u` is the user's position in the user list, `HASH_TIME` is either ~0.021 or ~0.027 seconds, and `PAD_MILL` is an unknown time between `0` and `1.001` seconds. The login time for target user `t` is `L_t = PAD_MILL + p_t * HASH_TIME`, and for the attacker is `L_a = PAD_MILL + p_a * HASH_TIME`, so `delta_t = L_t - L_a = (p_t - p_a) * HASH_TIME`.

(As noted above, `PAD_MILL` is set randomly, but is not changed after the startup of the Phonemaster server, so it is the same in `L_t` and `L_a`.) Calculating `delta_t / HASH_TIME` gives `p_t - p_a` and as `p_a` is known (as it is returned by the server after the attacker's Active Operation request for the Phonemaster userlist) `p_t` can be determined, thus revealing the target user's username."

Evaluation:

Correct; they identified the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.12 Powerstate

A.7.1.6.2.12.1 Question 031 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"No, there is not algorithmic complexity vulnerability in space that would cause the challenge program's memory usage to exceed the resource usage limit given the input budget. The primary source of memory usage in the challenge program is the creation and manipulation of large matrices in the `while` loop in `PowerStateSolver::solve()`. Code inspection and profiling revealed that the largest matrix created in this process is `H`. The dimensions of this matrix, as of many of the others in `solve()` are determined by user input. `H` has a width of `measurements.size()` and a height of `this.nbus.intValue() * 2 - 1`. `nbus` is the number of buses in the submitted power network file. This number of buses is capped by `State::verity()` to 1000, so the height of `H` is capped at 1999. From `State::generate_measurements()` we learn that `Measurement`s are created approximately one or two per bus and two per line. Again per `verity()` see that the number of lines is capped at 10000. Thus the width of `H` is capped at approximately 12000. However, this limit is not achievable in practice, as each bus and line requires, at minimum, approximately 30-50 bytes of input, and so at most 1-2000 buses and lines combined can be submitted within the input budget. […]"

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.12.2 Question 037 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes. […] example output:

% Total % Received % Xferd Average Speed Time Time Time Current

Dload Upload Total Spent Left Speed

100 4613 0 0 100 4613 0 3 0:25:37 0:20:26 0:05:11 0Bus: 1 Voltage: NaN Phase Angle: 0.0

Bus: 2 Voltage: NaN Phase Angle: NaN

Bus: 3 Voltage: NaN Phase Angle: NaN

Bus: 4 Voltage: NaN Phase Angle: NaN

Bus: 5 Voltage: NaN Phase Angle: NaN

Bus: 6 Voltage: NaN Phase Angle: NaN

Bus: 7 Voltage: NaN Phase Angle: NaN

Bus: 8 Voltage: NaN Phase Angle: NaN

Bus: 9 Voltage: NaN Phase Angle: NaN

Bus: 10 Voltage: NaN Phase Angle: NaN

Bus: 11 Voltage: NaN Phase Angle: NaN

Bus: 12 Voltage: NaN Phase Angle: NaN

Bus: 13 Voltage: NaN Phase Angle: NaN

Bus: 14 Voltage: NaN Phase Angle: NaN

100 5150 0 537 100 4613 0 3 0:25:37 0:20:26 0:05:11 0

In PowerStateSolver.solve() there is a while loop that runs for as long as the "tol" variable is greater than one. We observed that setting the initial voltage at one of the buses to be large (9E29), causes tol to increase from its initial value of 100 with each iteration. When the challenge program eventually terminates, it does so when tol overwrites to NaN, after hundreds of thousands of loop iterations. As indicated in the curl output above, our request only uses a small fraction of the input budget (about 5kB).”

Evaluation:

PoV confirmed

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.13 PSA

A.7.1.6.2.13.1 Question 010 (AC Time, Intended Vulnerable, Answered: No)

Response:

"No, there is no algorithmic complexity vulnerability in time that would cause the response time of the benign request to exceed the resource limit given the input budget. While PSA performs a considerable amount on the input program the input budget is tight enough (especially after the preprocessing done on the client to transform the input from raw Python code to the JSON representation required by PSA) that only a small and simple Python program can be sent. Exceeding the resource limit would thus require inducing either a very deep (if not infinite) recursion or infinite loop in PSA with the input Python program, but an investigation of the various analysis modules in PSA did not reveal any such recursion or loop."

Evaluation:

Incorrect; they failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.13.2 Question 030 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

"No, there is not algorithmic complexity vulnerability in space that would cause the challenge program to store files with combined logical sizes that exceed the resource usage limit given the input budget. PSA does not, as part of its processing, store any considerable volume or quantity of files. The input budget is far too small to allow for sending multiple requests are thereby accumulate large numbers of small files, so the only possibility is to induce the storage of 20 MB or more of files via one request, but, as with Question 010, the overhead of the JSON conversion from Python means that only a very very small and simple program can be submitted to PSA. Based on examination of the code and empirical observations there is little to no blow-up from the submitted JSON to the files stored by PSA; all stored data structures are linear or approximately so in the size of the inputs with small coefficients. Zip bombs and other similar compression exploitation techniques were investigated but the Zip file handling in PSA contains several safeguards and limitations that close off this avenue of attack."

Evaluation:

Incorrect; they failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.14 Roulette 1

A.7.1.6.2.14.1 Question 013 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"No. Calculating the winning slot begins with selecting a random seed value in Game.getTime(). The getTime() function returns randomRotateTime.subsequentDouble(), which returns a uniform random value between -30 and 60 generated with a SecureRandom. This value gets passed through to RouletteWheel.rotate(), which calculates the winning number. In this function, the randomness is shifted, and the magnified by angualarRate, which is at least 100, resulting in a random value between -4300 and 4300. The winning slot is determined from this value and from the previous winning slot. Any winning slot is very near equally likely to be chosen, and there is nothing observable to the user while this calculation is taking place that would reveal any information about the initial random value chosen, or the resulting winning slot. We conclude that an attacker cannot discover the winning number before betting is closed in a round."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.14.2 Question 019 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"No. When Roulette2 is run by the provided startServer.sh script, it writes to a single database file named playersDB.ser in the data directory. The file size of the database is initialized to roughly 2.1 MB and is sparsely populated. Once a Participant is connected, the Participant's data is serialized and committed to the database.

The database is driven by the external library MapDB, and uses a hashed tree map (HTreeMap) to store data with key-value pairs. In this case, keys are strings, and values are the serialized Participant object. A Participant consists of three components:

1) The UTF string representation of the user id, and public key modulus and exponent value concatenated as one
2) Fund amount
3) Connection status

The database stores each serialized Participant sequentially, filling up the sparse data as the file remains 2.1 MB in size. When a Participant is updated in the database, the player information is updated with current values and a new entry is not created. As each Participant object is saved by a key, the corresponded serialized data is overwritten. To grow the database, many new players must be added which can not be done with the 10 kB input budget given."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.14.3 Question 048 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"No. The packet containing the bet information is sent in the following format:

{ 6 bytes + bet_type_in_ascii + 2 bytes + bet_value_in_ascii + 1 byte + bet_amount }.

Based off the ASCII lengths of the possible bet types and values, and assuming a one byte value for the bet amount, the packets are of the following sizes:

17 bytes = number (single digit)

18 bytes = number (double digit) or color (red)

19 bytes = parity (odd)

20 bytes = color (black), dozen (first/third), parity (even)

21 bytes = dozen (second)

For each bet type/value, to correctly guess the amount (again, assuming a 1 byte value) would take 256 oracle queries (since 1 byte corresponds to the values 0-255 inclusive). Therefore, based on packet size it would take the following amount of oracle queries.

17 bytes = number (single digit) == 256 queries

18 bytes = number (double digit) or color (red) == 256*2 queries

19 bytes = parity (odd) == 256 queries

20 bytes = color (black), dozen (first/third), parity (even) == 256*3 queries

21 bytes = dozen (second) == 256 queries

Even with the simplification that the bet amount is only 1 byte (which only covers bet amounts up to $255), we are already over the input budget."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.14.4 Question 054 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"No. Users have a limited number of options for interacting with the Roulette server. We analyzed each of the available commands and did not find any way for an attacker to achieve the stated goal. Neither connect nor disconnect requests seem to have any effect on CPU usage and complete very quickly. Bankroll requests are quickly looked up and responded to after a 3 millisecond delay. We did not find any way to corrupt the lookup tables or interrupt the scheduled response. When placing a bet, the wager is created in a straightforward manner, and then added to the BetList.

There was just one point of interest we identified in these processes: BetList.wagers is an ArrayBlockingQueue with capacity Quantities.capacity and BetList.incorporateWager() calls this.wagers.offer() with a timeout of 120 seconds. This means that if the bet list were filled beyond capacity, the request would block for 2 minutes, which would also block another user's bankroll command for over 90 seconds as desired. However, Quantities.capacity = 10000, and we have an input budget of only 10 kB. We cannot send 10000 wagers win 10000 bytes of input. There is a line in the DozenWager constructor which divides Qunatities.capacity by 12 if it is greater than 1000, however any invocations of this constructor come after the initialization of the BetList, and the BetList is never recreated while the server is running, so the ArrayBlockingQueue in BetList always has a capacity of 10000. We conclude that there is no AC vulnerability in time that would cause the response time of a bankroll command to exceed 90 seconds.

We note that an AC vulnerability in time does exist in roulette_2, although there is no question asking about it. The attack is against the same blocking queue analyzed here, but the queue capacity is only 100, making it feasible to fill the queue with wagers while staying within the input budget."

Evaluation:

They answer no for this question (correctly), but note that there is a separate AC vuln that is not asked about.

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.15 Roulette 2

A.7.1.6.2.15.1 Question 011 (SC Space/Time, Intended Vulnerable, Answered: Yes)

Response:

"Yes. Timing when a bid reminder is sent out and knowing the previous winning slot are enough to determine the next winning value before it is announced. An attacker can activate bid reminders by betting an amount larger than their bankroll. This enable the attacker to learn the value of wagerReminderTime with at least 10 seconds to spare before bets are closed. The wagerReminderTime is stored in lastTime, which is used in the call to Game.takeTime(). The takeTime() function returns lastTime plus randomSpinTime.ensuingDouble, but the values from GameParameters used to initialize randomSpinTime, 0.01 and 0, are too small. As a result, ensuingDouble returns a uniform random value between -0.01 and 0.01. The takeTime() value is used as the spinTime in Croupier.spinTheWheel(), which determines the winning slot via RouletteWheel.spin(). In the spin() method, this.angularSpeed is multiplied by the input time, but the angular speed is set to a value less than 1. This actually narrows the range of uncertainty due to randomness to less than the initial 0.02 width band. The only other value which is used to determine the winning slot is the previous slot, which is announced at the end of the previous wheel spin. An attacker can view the previous winning slot, time how long it takes for the wager reminder to be delivered, and then use those two values to calculate the next winning slot before it is announced. The attacker need one operation to view the previous winning slot, and one to submit a bid to turn on wager reminders. One second resolution is all that is required for the timing data. While the attacker should be able to guess the correct slot on the first try, there are additional oracle queries which can be used to guess adjacent slots if necessary."

Evaluation:

Correct; they identified the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.15.2 Question 033 (SC Space, Intended Vulnerable, Answered: No)

Response:

"No. The packet containing the bet information is sent in the following format:

{ 6 bytes + bet_type_in_ascii + 2 bytes + bet_value_in_ascii + 1 byte + bet_amount }.

Based off the ASCII lengths of the possible bet types and values, and assuming a one byte value for the bet amount, the packets are of the following sizes:

17 bytes = number (single digit)

18 bytes = number (double digit) or color (red)

19 bytes = parity (odd)

20 bytes = color (black), dozen (first/third), parity (even)

21 bytes = dozen (second)

For each bet type/value, to correctly guess the amount (again, assuming a 1 byte value) would take 256 oracle queries (since 1 byte corresponds to the values 0-255 inclusive). Therefore, based on packet size it would take the following amount of oracle queries.

17 bytes = number (single digit) == 256 queries

18 bytes = number (double digit) or color (red) == 256*2 queries

19 bytes = parity (odd) == 256 queries

20 bytes = color (black), dozen (first/third), parity (even) == 256*3 queries

21 bytes = dozen (second) == 256 queries

Even with the simplification that the bet amount is only 1 byte (which only covers bet amounts up to $255), we are already over the input budget."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.15.3 Question 046 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"No. When Roulette2 is run by the provided startServer.sh script, it

writes to a single database file named playersDB.ser in the data

directory. The file size of the database is initialized to roughly 2.1

MB and is sparsely populated. Once a Player is connected, the Player's

data is serialized and committed to the database.

The database is driven by the external library MapDB, and uses a hashed tree map (HTreeMap) to store data with key-value pairs. In this case, keys are strings, and values are the serialized Player object. A Player consists of three components:

1) The UTF string representation of the user id, and public key modulus and exponent value concatenated as one
2) Fund amount
3) Connection status

The database stores each serialized Player sequentially, filling up the sparse data as the file remains 2.1 MB in size. When a Player is updated in the database, the player information is updated with current values and a new entry is not created. As each Player object is saved by a key, the corresponded serialized data is overwritten. To grow the database, several new players must be added which can not be done with the 10 kB input budget given."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.16 Securgate

A.7.1.6.2.16.1 Question 008 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"Yes. An attacker can view the responses to image block requests and use the sizes of those responses to determine which license plate image was sent to the client, and thus which license plate number the client sent to the server.

The client requests a license plate from the server and then requests pieces of the image (referred to as Image Blocks) by using the REST URL GET /get_image_block?id=<id>&token=<token>. License plates have different color schemes, and we can identify different colors by checking the length of the returned image blocks. Different color image blocks have different sized ciphertexts, and there is no padding to make the packets uniform size. For a

single plate image, There are 7875 image blocks sent in separate TCP sessions. The attacker has enough operations to view the lengths of all of the image blocks. Because the attacker has access to the plates directory, they can compute offline the sizes of image blocks sent and use this as a fingerprint for the observed image. There are no identical images so there is not a need for oracle queries.

The attacker actually does not need to do offline work to carry out the attack. It is possible to use the relative sizes of image blocks to identify the two different colors, and then print a rough image of the plate. This provides an outline of the letters and numbers on the plate which is clear enough to read. We have provided an exploit script which, given a packet capture of responses to the image block requests, prints out a representation of the plate."

Evaluation:

Confirmed their exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.16.2 Question 018 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"No. The attacker has a very limited set of endpoints with which to interact with the server. A GET request to "/get_next_plate" meremly returns a random plate to the user, and there is no significant computation. A GET request to "/get_image_block" gets the next block of the chosen plate, but these image blocks come from the given set of images and there does not appear to be a way to augment or add a custom image to that set of images. A POST request to "/transcribe_image" performs an AES/GCM decryption of a submitted encrypted plate and then writes to a log file. The decryption is done with Java libraries that are out of scope. The log write is only interesting becuase it is synchronized, but the input budget prevents us from sending more than a dozen requests (because of the TLS overhead)."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.17 Suspicion 1

A.7.1.6.2.17.1 Question 020 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"No. From manual inspection we identified MorseCode.decode() as the only method which could potentially store a file larger than the input budget. The decode() method creates a temporary file and calls the unzip() method, which writes decoded bytes to the file. In unzip(), there is a counter of bytes written so far (sumQuantity) and the method throws an IOException if sumQuantity > 1000000. Unless we can escape this check, the maximum size of the output file is capped at 1 MB. The only way to circumvent this check would be to reduce sumQuantity by putting a negative integer value in the stream. However, recordQuantity() throws an exception if sumQuantity < currentSum, which is the previous value of sumQuantity. This means the check will fail if a negative number is passed in and sumQuantity is less than it was the previous iteration. We conclude that we cannot cause the file to exceed 1 MB in size."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.17.2 Question 026 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"Yes. An attacker assigned as a spy can always determine the location before it is announced. The length of the first 3 answer announcement packets leaks the index of the location in the locations list. The DistrustCompetition.handlePointer() method loads the base 3 encoding of the location index into the first 3 bytes of DistrustCompetition.encoding. When the game host receives and forwards an answer to all other game participants via sendAll(), this method calls grabLength() to set the length of the message to be sent out. The grabLength() function returns HIGHEST_MESSAGE_QUANTITY + this.encoding[this.rounds]. So for rounds 0-2, subtracting HIGHEST_MESSAGE_QUANTITY = 1000 from the length of the answer announcement message reveals the one digit of the base 3 encoding of the location index. After seeing all 3 of these answer announcement messages, the spy can concatenate the 3 values and convert from base 3 to base 10.

The spy can then look at that index in the location list to find the location. The spy is guaranteed to see this information because the game enforces a minimum of 3 rounds of questions and answers before the game can end. The spy can find the length of the received message by printing out the value of msg.length in DistrustSecretary.handleReceivedMessage."

Evaluation:

Sounds very close to the intended; confirmed

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.17.3 Question 040 (SC Space/Time, Intended Vulnerable, Answered: No)

Response:

"No. There is no side channel in space and time from which the spy can discover the secret password. Passwords are checked in DistrustCompetition#isPasswordCorrect. The method does not bail early if the password is not correct. When a game is started, a packet is sent to everyone in the game that is dependent on the lengths of the following values: names of players in the game, location index, and password. An attacker knows the names of all players in the game, as well as the length of the location index (see location side channel in question 40). Therefore, the length of the password can be determined. However, the password can contain any character and be a max length of 5. Even limiting the password combinations to any letter upper or lower case is significantly over input budget to brute force the password with oracle queries. (52^5 == 380204032)."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.18 Suspicion 2

A.7.1.6.2.18.1 Question 022 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"No. Passwords are checked in DoubtTournament#isPasswordCorrect. The method does not bail early if the password is not correct. When a game is started, a packet is sent to everyone in the game that is dependent on the lengths of the following values: names of players in the game, location index, and password. An attacker knows the names of all players in the game, and knows that the location index has length 1 or 2 (since the highest location index only has

two digits). The length of the password can be determined within 1. However, even if the attacker knew the exact password length, there is insufficient information to determine the password within the input budget. The password can contain any character and be a max length of 5. Even limiting the password combinations to any letter upper or lower case is significantly over input budget to brute force the password with oracle queries. (52^5 == 380204032”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.18.2 Question 044 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

“Yes. MorseCode.decode creates a temporary file which is written to by the MorseCode.expand() method. At first glance, there appears to be a limit on the size of the data that gets written, as expand() tracks the amount of bytes written in aggregateContent and throws an exception if aggregateContent > 1000000. This would typically prevent the size of the file from growing beyond 1 MB. However, a user can pass in a negative integer in inStream, which will write 0 bytes, but will reduce the count in aggregateContent. Submitting negative values allows a user to prevent aggregateContent from exceeding 1000000 while writing an unlimited number of bytes to the temporary file.

Here is one byte array which writes 2.1 GB of '/' characters to a temporary file when passed to MorseCode.decode():

new byte[] {(byte) 0x2F, (byte) 0x80, (byte) 0x00, (byte) 0x00, (byte) 0x00, (byte) 0x2F, (byte) 0x7d, (byte) 0x2b, (byte) 0x75, (byte) 0x00}

The first 5 bytes result in a call to writeRepeated('/', -2147483648) and reduce aggregateContent to -2147483648. The second 5 bytes add 2100000000 to aggregateContent, which will still be negative. Since aggregateContent is less than 1000000, writeRepeated('/', 2100000000) is called, which writes 2100000000 '/' characters to the temporary file.

An attacker can trigger this behavior by sending a QUESTION or ANSWER message

to the game host with the malicious byte array above submitted in the Question.question or Answer.answer fields, which are passed to MorseCode.decode(). On the NUC, the file is written to /tmp/susp*. It takes a few seconds for the file write to complete as it is so large, and it can get auto-deleted after it is written, but the full 2.1 GB will be written."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.18.3 Question 057 (Null SC Space, Intended Not Vulnerable, Answered: No)

Response:

"No. The spy cannot always determine the location before it is announced with at least 95% probability of success within the input budget. As in Question_026, the location index is placed in the first 3 bytes of DoubtTournament.encoding in base 3. On answer announcements, the host calls transferAll, which determines the length of the sent message via grabMagnitude(). The grabMagnitude() method returns HIGHEST_MESSAGE_CONTENT + thus.encoding[this.plays]. If this.plays were in fact the number of question and answer rounds as one would expect then the spy would always be able to determine the correct location. However, this.plays gets incremented any time getPlays() is called, which happens at least twice per round of questions. So in a typ"ical game, the only meaningful information leaked to the spy about the location would be the value of the digit in the 3s place of the base 3 location index. This would allow the spy to narrow the search to a maximum of 9 possible locations. We note, however, that getPlays() is also called when any user attempts to guess the identity of the spy, or guess the location. In the worst case, a player could submit a spy or location guess before any questions have been answered, which would raise this.plays above 2. As a result, subsequent answer requests would never reveal the contents of encoding[0], encoding[1], or encoding[2]. The spy in this case receives no information about the location except by playing the game and learning from player answers."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.18.4 Question 060 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

" Yes. When a send_question or send_answer message is sent by any user, it is morse code encoded, and then compressed. This compression takes the form of <value><n>, where <n> is the number of times that <value> appears consecutively (MorseCode#compress). When the message is decompressed, for each <value><n> pair in the compressed stream, it writes <value> <n> times to a file (MorseCode#expand). It then proceeds to read this file in its entirety into memory (MorseCode#decode). The expand() function does check to ensure that it does not read in more than 1000000 bytes, however this is not enough to prevent an attack. An attacker simply needs to replace the byte[] returned by compress() to {(byte) 0x2F, (byte) 0, (byte) 0x0F, (byte) 0x42, (byte) 0x40} (it is possible to debug break the program during execution to make this change). This is the byte 0x2F ("/" in ASCII) appearing 0x000F4240 (1000000) times.

When testing on a nuc, the malicious response caused the program to use 3.8GB of memory."

Evaluation:

Confirmed exploit for unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.19 ThermoMaster

A.7.1.6.2.19.1 Question 021 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"No. The server maintains a log file, and writes 32k lines of data to this file on a standard run. However, the overall size of this file is under 1MB. The log-writing mechanism is called every 1000 iterations of the PID controller loop (which iterates a fixed 32 million times). There are no other locations in the challenge program that write to this log file. An attacker can also upload files to the server, but we were unable to cause expansion on the server side to reach the resource target (e.g. via a decompression bomb)."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.19.2 Question 038 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"No. The simulation runs in constant time - PID#run completes a while loop 32,000,000 times for every simulation. Enabling/disabling integral,

proportional, derivative, or windup options do not have an effect on runtime. The simulation only runs if 'set current' and 'set setpoint' are sent in a request, so issuing a second request after the target request resets the data that the attacker is trying to discover (and also does not have a timing difference anyways)."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.19.3 Question 052 (AC Time, Intended Vulnerable, Answered: No)

Response:

"No. The challenge program is small (only 6 classes) and we found no code outside of the PID class that could potentially have an AC time effect. The PID controller logic in the run() method uses a fixed number of iterations, and the "set proportional constant", "set derivative constant" and "set integral constant" commands are nonfunctional. This essentially leaves the attacker with only the base and target temperatures, changing from Fahrenheit to Celsius and vice versa, and showing the state before execution. While switching from Celsius to Fahrenheit can allow the user to exceed the intended maximum temperature, this has little effect to the computation time (again because of the fixed number of iterations in main PID loop). The other operations occur nearly instantly."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.20 Wordshare

A.7.1.6.2.20.1 Question 002 (SC Space, Unintended Vulnerable, Answered: Yes)

Response:

"Yes. A non-privileged user is allowed to view and add secure documents. An attacker can use a get request to retrieve the target document, which will contain the base64 encoding of the secure word and the chosen IV. The attacker can resubmit the same secure word multiple times with different values for the IV. Varying the IV changes the values of the local variable

decrypt in WordShareDocControl.addFull(). If the bytes of decrypt fall outside the range 48-90, then the application attempts to fill in the value from a dictionary encoding defined in the added document. If any byte in decrypt has value 32 (the space character), then the number of bytes written to the byte buffer from bb.put(copyOfRange) will be reduced. Typically, a user would have no insight into how many bytes have been written to this buffer. However, the buffer has a maximum size of 10000 characters. An attacker can identify whether or not the number of bytes written was reduced by pre-filling the buffer nearly to capacity, picking a value such that if there is no space in decrypt, then the buffer will be filled beyond capacity, but if there is a space, fewer bytes will be written and the buffer will not exceeed its capacity. The attacker can then check whether the length of the response matches 'err:null' indicating a buffer overflow exception, or 'res:Ok' indicating the file was successfully added. By using a dictionary with space characters in either the first or second position, and a carefully chosen iv, the attacker can determine whether a specific byte of decrypt falls within a given byte range. A binary search on each byte is possible to identify specific bytes in the plaintext of a secret word.

We have implemented this attack in an exploit script which accepts the name of the target document and outputs the plaintext of the secret word. The script only works for characters between 48 and 90. We need at most 6 operations to binary search these 43 characters, and the maximum length of a secret word is 16. Our operational budget is 6*16 = 96 in the worst case.

Our attack can be extended to all ascii characters using a similar binary search technique. Updating our script to support all such characters presented some annoying edge cases that made it not worth the development effort, however it is possible to identify characters outside the 48-90 range within the input budget. Some of the edge cases include spaces early in words changing the number of bytes written, underscores at the end of words changing the number of bytes written, leading spaces in the first entry of the encoding dictionary being ignored, needing to have spaces in entries of the encoding dictionary in order to avoid

failing the word length check, and multi-character dictionary encodings causing adjacent characters to be skipped over in the decryption process. A rough estimate for the number of operations to resolve these issues is an overhead of 16 operations per character (to adjust the initial fill length of our buffer), plus 7 operations to binary search each character. This would cost up to a total of 16*16 + 7*16 = 368 operations, which is still less than half of our input budget."

Evaluation:

Confirmed their exploit for unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.20.2 Question 005 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

"No. The WriteDoc method is the only place where any file writes occur, so we limited our analysis to that method. The only promising avenue to give the needed "blowup factor" for the 35 MB file write is using the "#" encodings that allow for expansion in your uploaded documents. However, those expansions only get stored in memory in the T9Trie, not in the file that gets written to disk. The file that gets written to disk is exactly the file that you upload, with no extra processing influencing the file write. Because of the lack of "blowup factor", we conclude that there is no algorithmic complexity vulnerability in space."

Evaluation:

Correct after review of question

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.20.3 Question 047 (SC Time, Unintended Non-Vulnerable, Answered: No)

Response:

"No. In our analysis, we determined that the search functionality is all that would be useful for our active operations. Because search is performed on prefixes in the trie, we had investigated whether timing differences could enable a search similar to a binary search. However, during testing and performing timing of various search operations (such as prefixes that match vs. prefixes that don't match, various lengths that search at different depths of the trie, etc.), the timing of the find() method was very consistent. This indicated that there is no information leak in this method.

Additionally, even if there was such an information leak, it would be impossible to distinguish secret words and non-secret words. This is because they exist in the same trie, with no distinguishing features or annotations in the trie.

We also note that an attack such as the one in Question 002 is not possible because in this question we do not have access to any secure document. There is no mechanism for an attacker to enumerate all documents or otherwise retrieve a secure document on the server."

Evaluation:

Correct. Two Six observed that incorrect wording of the question created a trivially non-vulnerable challenge.

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.21 YapMaster

A.7.1.6.2.21.1 Question 015 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

"No. Each user has a "[user]_yapmap.map" file that is roughly 26 kB, but is capable of growing. Upon manual analysis of code, this seemed to be the only promising file write for this question.

The various "MemoryMap" classes provide an interface to these files. The method MemoryMapWriter.put() writes to these files. Notably in this method, the argument "id" impacts where in the file the data buffer will be written. If it is large, the buffer will be written at a large offset in the file. In order for the file to become 100MB as required by the question, "id" will need to be 781,000 or greater.

Unfortunately, while the id can be influenced by user input, there are several conditions and checks that prevent the id variable from ever becoming large enough. By manual code analysis, there does not seem to be a mechanism to bypass these checks. Therefore, we conclude that there is no algorithmic complexity vulnerability in space."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.1.6.2.21.2 Question 017 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"No. We can observe the network traffic throughout the entire duration the simulateUsers.sh script is run for side channels in time. However while observing several different users add (POST) and get (GET) yap messages, no significant findings in time were found. When a message is encrypted using the POST API, the client performs all the encryption and then sends the data to the server. Similarly for the GET API, when supplied the correct user and message ID, the encrypted text is sent from the server to the client for decryption, minimizing any network traffic delays from observation.[…] Given that we have the encrypted cipher text of the most recent yap message and the initialization vector used for encryption, to decrypt the data we still need to discover the encryption key. Keys used for encrypting messages are 16 bytes in length and are never reused, requiring us to brute force all possible keys. Since the yap server does not have a mechanism to submit an arbitrary encrypted yap message and gain information on whether decryption is successful, or a mechanism to reduce the space of possible encryption keys to a more tractable number. We conclude that obtaining the plain text message can only be done offline by brute forcing the encryption key."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.1.6.2.21.3 Question 042 (SC Space, Intended Vulnerable, Answered: No)

Response:

"No. While observing the network traffic during the execution of the simulateUsers.sh script, we were able to view several different users login and add (POST) and get (GET) yap messages. Before a message is posted to the Yap server by an authenticated user, the message and label are first encrypted using the RC4 cryptographic cipher and a uniquely generated key. This encrypted message is then sent to the server along with the random

initialization vector, HMAC, message id, user and yap alias, and an additional encrypted value named 'encrypted'; all of which can be intercepted over HTTP. The additional data name 'encrypted' is the plain text of the user's password encrypted with AES-CBC using the user's corresponding decode key and the same random initialization vector as the RC4 encrypted yap message. Although this encrypted password is sent for every yap message post, it is never used by the server unless the client is attempting to login or modify a yap message. […] After identifying the most recent yap, we can either attempt to decrypt the message by brute forcing the key or attempt to falsely share the message to the YapMaster user we have credentials for. […] Since the yap server does not have a mechanism to submit an arbitrary encrypted yap message and gain information on whether decryption is successful, or a mechanism to reduce the space of possible encryption keys to a more tractable number. We conclude that obtaining the plain text message can only be done offline by brute forcing the encryption key."

Evaluation:

Incorrect; they failed to identify the intended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.2 Utah

A.7.2.1 Answer Statistics

Table A-155: Engagement 7 Utah Question Accuracy

Question Type	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
SC Space	10	6	60
SC Time	11	7	64
SC Space/ Time	7	5	71
AC in Space	17	12	71
AC in Time	9	7	78
Total	**54**	**37**	**69**

Table A-156: Engagement 7 Utah Vulnerability Accuracy

Vulnerable/ Response	Number Attempted	Reviewed Number Correct	Reviewed Accuracy (%)
Vulnerable	27	10	37
Not Vulnerable	27	27	100
Yes Answer	13	10	77
No Answer	41	27	67

A.7.2.2 Utah Specific Responses

A.7.2.2.1 Calculator 3

A.7.2.2.1.1 Question 023 (AC Space, Unintended Vulnerable, Answered: No)

Response:

“The only freely growing data object in memory is the "dataInProgress" list stored in reformulate. However, this is only added to if the "complex" static variable is set to true, which in turn only happens if "dataIntake" is called, which is is not.”

Evaluation:

Was originally correct but another team found an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.1.2 Question 025 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

“The only files written are written by the error logger. It is possible to make it print (sizeable) stack overflows. However, a stack overflow is only ~80kb, meaning it takes ~25k of them to reach the target. Given a budget of 60kb, this requires multiple stack overflowers from a single request.

However, each stack overflow exceptions up to the top level, causing that section to hang. The only catch blocks that could catch one beforehand cannot generate stack overflows beneath them, and/or are in unreachable code or have catch blocks that throw other "lethal" errors.”

Evaluation:

A

Post-Engagement Analysis Ruling: Correct

A.7.2.2.1.3 Question 034 (AC Time, Intended Vulnerable, Answered: No)

Response:

“In the math code, he following methods have loops that can delay the outputs.

(1) Calculator.handleOutput: […]
(2) LargeNumeral.Divide:[…]
Limitations on lengths of variables for roots and powers don't allow for big delays.
Outside of the math code, the parsing function is inefficient with regard to parenthesis. Deeply nested parentheses (like `(((((((((1)))))))))') slow it down, but not enough. Too many parenthesis causes a stack overflow and makes the current client hang waiting for a response, but this does not impact future requests”

Evaluation:

Incorrect; they failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.2 Calculator 4

A.7.2.2.2.1 Question 036 (AC Space, Intended Vulnerable, Answered: Yes)

Response:

“The "Modify" class contains an implementation of (I think) Fast Fourrier transform. For unclear reasons it includes a static variable "arrayInProgress", to which intermediate values are added repeatedly during the recursive calls in a superlinear manner. […]
Putting all of it together: "9"*16384 + "*(10^512*10^512)" causes the list to explode in size to 400 million, which is more than enough to hit the target 4GB memory budget.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.2.2 Question 043 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

“In the math code, the following methods have loops that can delay the outputs.
(1) Calculator.handleOutput: […]

(2) LargeNumeral.Divide:[…]
Limitations on lengths of variables for roots and powers don't allow for big delays.
Outside of the math code, the parsing function is inefficient with regard to parenthesis. Deeply nested parentheses (like `(((((((((1)))))))))') slow it down, but not enough. Too many parenthesis causes a stack overflow and makes the current client hang waiting for a response, but this does not impact future requests”

Evaluation:

Correctly determined it was not vulnerable.

Post-Engagement Analysis Ruling: Correct

A.7.2.2.2.3 Question 045 (AC Space, Intended Vulnerable, Answered: No)

Response:

“The only files written are written by the error logger. It is possible to make it print (sizeable) stack overflows. However, a stack overflow is only ~80kb, meaning it takes ~25k of them to reach the target. Given a budget of 60kb, this requires multiple stack overflowers from a single request.

However, each stack overflow exceptions up to the top level, causing that section to hang. The only catch blocks that could catch one beforehand cannot generate stack overflows beneath them, and/or are in unreachable code or have catch blocks that throw other "lethal" errors.”

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.3 CyberWallet 1

A.7.2.2.3.1 Question 006 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

“Looking at the paths that the application takes, there is no algorithmic vulnerability in space that the attacker can exploit within the input budget. Since the attacker cannot look at the data folder where all the data for the bank,clients and accounts come from there is

no way to blow up the data structures that store these objects. Also with server configurations, we didn't find anything vulnerable other than the OptimizedTimeser class that creates a lot of BigIntegers which could occupy more memory. But this method is not called since the application does not use the KeyExchangeServer. […]"

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.3.2 Question 035 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"Each has a single "sponsored content" gif, the size of which dominates the total page size.

Sponsored content on the "Account Summary" page consists of a single gif. This comes from the AccountInfoManager class, which picks the (tens value ofthe) index of the gif to be shown based on log(account balance + random), where random follows a gausian distribution and is likely too small to matter. Effectively, this means that the number of digits in the account balance causes the gif to be chosen from a particular set of 5; this presents a side channel in size.

A check of what indexes are generated based on each account value shows: essentially, the value passed absolutely shows the account tier. […] This value limits the selected gif to be one of a set of images. As the images all have different sizes, and the size of the images dominates the rest of the page, it is possible to observe the size of the transmitted data to tell which image was sent, and in turn what set that image was picked from, and in turn the account tier."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.3.3 Question 051 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“There is no side channel vulnerability in time from which an attacker can find information to transfer money from taget user's account to own account. One path where the attacker can interact with the application is the login page where the target user enters login credentials. The LoginManager that checks for username and password does not provide any vulnerable side channel in time for the attacker to guess the username/password. Also, the username and password are not exposed anywhere else in the webpage. This rules out the possiblity of gaining access to a target user's account.

The other path where the attacker can interact with the application is through the transfer money utility. We found a way here to steal the money by making the authorizationNumber field negative that is being used in ExchangeCore.grabQuantity() that determines the amount of money that needs to be transferred. Making the quantity negative would transfer the amount from target account to attacker's account. But this is possible only if we know the target user's account number. VerificationManager.makeTransfer checks if the given account number is present in the bank repository's hashset and gives out an error message if it is not a valid account. Hence there is no side channel here to try and figure out the target user's account number.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.4 CyberWallet 2

A.7.2.2.4.1 Question 024 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

“Cyberwallet 1 and 2 have very similar code which chooses an ad semi-randomly based on account value. […] This does show a side channel in space, as the probabilities of different ads being picked depends on worth. However, multiplying by the gaussian distribution introduces a large amount of error, which is not entirely obviated by the Math.log10 call. […]

While this clearly shows a dependency, it is not sufficient to achieve the required confidence in only 2 operations."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.4.2 Question 059 (SC Time, Intended Vulnerable, Answered: Yes)

Response:

"There is a side channel vulnerability in time from which an attacker can steal money from target user's account into their own account. For transferring money we would need the routing number and target account number. We can use the attacker's routing number and try to steal money from an internal bank account. Next step is to figure out a target user's account number. AccountConductor.insertAccountFace() stores all the account numbers in a Trie. Whenever a money transfer is made, AccountConductor.findAccountFace() searches this Trie to find if the destination account number is a valid one. FindAccountFace() calls testForDuplicateError() for every digit it checks to identify if there is duplicate account number. This is an expensive call that searches through the list of all 8000 accounts to look for duplicates hence providing us a side channel in time. Since the account number can only contain numbers from 0-9 and it has only 10 digits, we can device the following attack.

Start with an account number with all 0's and try to make a money transfer from attacker account. If the account number is wrong, we get a validation error. Calculate the time it takes to return this error. This time denotes the number of digits that were correct before it encountered a incorrect digit as the digits in the account number are validated starting from left to right. So, we fix the correct digits now in the account number and start trying out values from 0 until 9 for the next incorrect digit until we identify the correct value for that digit. Repeat this for all the digits. This way we can figure out all the digits of the account number just by comparing the time it takes to validate. This can take a maximum of 100 operations in

worst case, since at the max we would try 10 values for digit 1 and fix it , then try 10 values for digit 2 and so on

With the account number that we found, we can do the following to steal the money from that account. In the money transfer flow of the application, TransactionValue.takeQuantity() returns the product of quantity and authorized variables. Authorized is simply the 5th field of the money transfer form, which is comma separated; the 4th field, description, lacks any input validation. We can thereby put the input description as ",-100", and receive 100X more money than you were checked as giving. (It is possible to steal more money than the target has. We checked that both attacker and attacked accounts are correctly modified.)"

Evaluation:

Confirmed exploit.

Post-Engagement Analysis Ruling: Correct

A.7.2.2.5 Door Master

A.7.2.2.5.1 Question 029 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"Vulnerabilities exist but no attack seems possible with the given input budget of 2KB. The request to create keyFobs contains two major paths: one when there is a valid connection to the remote and another where the connection is broken.

The first path that involves creating a key pair, storing the public key on the remote and the private key on the DoorServer does not expose a possibility to attack within the input budget.[…]

The second path also does not provide a window to attack since it just returns an error message that the connection to the client is broken. There are no retry mechanisms involved here to make an attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.5.2 Question 041 (SC Time, Intended Vulnerable, Answered: No)

Response:

“The attacker can get access to the key fob only if we gets hold of the encoded private key of the target user. The target user makes two requests to the DoorServer.
The first request is to create a keyfob. The server creates this keyfob randomly without taking in any input from the user and returns the encoded private key of the fob back to the user. The attacker cannot possibly determine the private key just by the time it takes to generate the keyfob since there is no time pattern from the responses and the length of the responses are all the same. Also since there are no user inputs to generate the keyfob, the attacker cannot correlate input to response time here.
The second request is where the target user uses the encoded private key to get authorization to the door. […] These calculations however cannot be exploited to identify the private key since the time difference for different keys would be negligible.

Evaluation:

Incorrect; they did not identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.6 EMU 6502

A.7.2.2.6.1 Question 004 (SC Time, Intended Not Vulnerable, Answered: N/A)

Response:

N/A

Evaluation:

N/A

Post-Engagement Analysis Ruling: <Correct/Incorrect>

A.7.2.2.6.2 Question 012 (SC Space, Unintended Vulnerable, Answered: N/A)

Response:

N/A

Evaluation:

N/A

Post-Engagement Analysis Ruling: <Correct/Incorrect>

A.7.2.2.6.3 Question 014 (SC Time, Intended Vulnerable, Answered: N/A)

Response:

N/A

Evaluation:

N/A

Post-Engagement Analysis Ruling: <Correct/Incorrect>

A.7.2.2.6.4 Question 016 (AC Time, Intended Not Vulnerable, Answered: N/A)

Response:

N/A

Evaluation:

N/A

Post-Engagement Analysis Ruling: <Correct/Incorrect>

A.7.2.2.7 InAndOut 1

A.7.2.2.7.1 Question 007 (AC Time, Unintended Vulnerable, Answered: Yes)

Response:

“The attack is very simple. First order a pizza and obtain a valid Validation Code. For example "2-0-4-0-6-32" then pick up a pizza but provide a code that can't exist but whose Binary Code is valid, for the example code " 9-0-7-0-3-17". Then the web server is going to loop forever waiting for a pizza that doesn't exist.”

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.2.2.7.2 Question 027 (AC Space, Intended Vulnerable, Answered: No)

Response:

“The attacker has two possible attack vectors (apart from login). The first is ordering a pizza and the second is picking up a pizza. The second is very limited, the Validation code is checked against a set of codes and not much else. The first is also very limited. There are only 14 possible toppings (0 is a valid integer but ignored by both the server and the inventory manager) and the maximum quantity of pizzas is 10.

Generating the validation code has nothing to do with the user so it can't be attacked.

Finally what remains is the inventory manager. The web server sends the

toppings and quantity to it as strings (and already checked the validity of almost everything). The Inventory recieves first an update to the stock of ingredients and then the actual order. For the update tit checks the ingredient index versus the stock. If there is not enough then the it simply adds more (depending on the popularity). For the actual order it stores a history but in the limited budget this can't be exploited."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.7.3 Question 049 (SC Time, Intended Vulnerable, Answered: No)

Response:

"There can never be more than 63 pizzas waiting for pick up. The Binary Code then determines which of the 6 integers of the validation code will be non-zero. The exact value of the non-zero integers is determined by another random (and convoluted process) which eventually generates one the possible values listed above. It is important to note that neither the Binary Code nor the Validation Code is influenced by what the user ordered.

The latest Validation Code, of that user, is stored in the users websession and also displayed on the confirmation page. It is never sent to the inventory server.

The attacker to pick up a pizza only needs to know the Validation Code of the victim. We do not believe it is possible to determine the code by monitoring the timing of the network traffic of the victims session. There is significant noise in several steps of the process. Most notably when generating Validation Code there is random noise introduced by the random generation process, sending the ingredients used to the inventory server and selecting the Binary Code. All of these contribute to it not being possible to determine the Validation Code in the provided budget."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.7.4 Question 055 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"The Binary Code then determines which of the 6 integers of the validation code will be non-zero. The exact value of the non-zero integers is determined by another random (and convoluted process) which eventually generates one the possible values listed above. It is important to note that neither the Binary Code nor the Validation Code is influenced by what the user ordered.

The latest Validation Code, of that user, is stored in the users websession and also displayed on the confirmation page. It is never sent to the inventory server.

The attacker to pick up a pizza only needs to know the Validation Code of the victim. We do not believe it is possible to determine the code by monitoring the space of the network traffic of the victim. Although the code is sent two times its limited length makes many codes use the same space. Learning the code thus simply isn't feasible in the budget that is provided."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.8 InAndOut 2

A.7.2.2.8.1 Question 028 (SC Time/Space, Intended Vulnerable, Answered: No)

Response:

"There can never be more than 63 pizzas waiting for pick up. The Binary Code then determines which of the 6 integers of the validation code will be non-zero. The exact value of the non-zero integers is determined by another random (and convoluted process) which eventually generates one the possible values listed above. It is important to note that neither the Binary Code nor the Validation Code is influenced by what the user ordered.

The latest Validation Code, of that user, is stored in the users websession and also displayed on the confirmation page. It is never

sent to the inventory server.

The attacker to pick up a pizza only needs to know the Validation Code of the victim. We do not believe it is possible to determine the code by monitoring the space and time of the network traffic of the victim. Although the code is sent two times its limited length makes many codes use the same space. Learning the code thus simply isn't feasible in the budget that is provided. The timing of the network traffic is very noisy. There is significant noise in several steps of the process. Most notably when generating Validation Code there is random noise introduced by the random generation process, sending the ingredients used to the inventory server and selecting the Binary Code. All of these contribute to it not being possible to determine the Validation Code in the provided budget."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.8.2 Question 039 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"If the Binary Code is valid then it checks if the Pizza is ready. To do so it calls ProcessedOrders.isReady which checks the Validation COde that was provided against the set of ready pizzas. If the pizza is not ready then the server waits for 1 second and tries again, and again, and again. And will not return until the pizza is ready.

The attack is very simple. First order a pizza and obtain a valid Validation Code. For example "2-0-4-0-6-32" then pick up a pizza but provide a code that can't exist but whose Binary Code is valid, for the example code " 9-0-7-0-3-17". Then the web server is going to loop forever waiting for a pizza that doesn't exist."

Evaluation:

Confirmed the exploit

Post-Engagement Analysis Ruling: Correct

A.7.2.2.8.3 Question 053 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

“There are only
14 possible toppings (0 is a valid integer but ignored by both the
server and the inventory manager) and the maximum quantity of pizzas is
10.
Generating the validation code has nothing to do with the user so it
can't be attacked.
Finally what remains is the inventory manager. The web server sends the
toppings and quantity to it as strings (and already checked the
validity of almost everything). The Inventory recieves first an update
to the stock of ingredients and then the actual order. For the update
tit checks the ingredient index versus the stock. If there is not
enough then the it simply adds more (depending on the popularity). For
the actual order it stores a history but in the limited budget this
can't be exploited.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.9 Litmedia 1

A.7.2.2.9.1 Question 009 (AC Space (Memory), Unintended Vulnerable, Answered: No)

Response:

“Our tool pointed to a few places where algorithmic attacks could be
possible. Further investigation revealed that none of these were
susceptible to attacks on the memory usage. Some of these are detailed
below.

The JSON classes are only used on server start-up, and so are not
relevant as they cannot be exploited at runtime. […] getTagCombinations(...) is a method which employs the use of some
mutually recursive helper methods. All together, these three methods
serve to produce all possible combinations of the 25 tags which are 1,
2, or 3 tags in length. This produces a list which is 25+300+2300 =
2625 combinations in length. The content of the resulting list is fixed
by the tags which are initialized at server start-up, and this method

is only ever called during server start-up […] The updatePicks(...) method is called only during the
grabPublishingFile(...), but it appears to be the most algorithmically
interesting method discovered in our analysis. The method is an
implementation of a sorting algorithm over the PickNodes. The sort is
an in-place, unstable, descending sort on the ArrayList of PickNodes in
the PublishingAtheneumActual. Because the sort is in-place, it does not
increase memory usage in the obvious way. The method does use a
recursive helper method, but the stack increases only logarithmically
with respect to the input size in the worst case. Since the input is
fixed at all points of execution (the length of the list of PickNodes
never changes), this sort algorithm cannot be exploited to consume
sufficient memory to meet the challenge question requirements."

Evaluation:

Was originally correct but another team discovered an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.9.2 Question 032 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"There are only two points when a user's preferences can become exposed
during regular server execution. One is when the user loads their home
page and the top picks are retrieved. The other is when a user clicks
on an article and their preferences are updated. […]
PublishingAtheneumActual.fetchTopPicks(...). This method
iterates over the user's PickNodes to find a specified number of nodes
with the greatest frequencies, but will only fetch 5 PickNodes due to a
hard-coded value in the HomePageGuide class. Therefore, this path of
execution does not appear to be exploitable. […] A user's preferences are updated every time they click on an article
(but at no other times). This triggers a call to
PublishingAtheneumActual.grabPublishingFile(...). In this method, the
user's PickNodes are iterated over to find the one containing the
designated PublishingFile's ID. At that point, the matching PickNode
will have its frequency incremented, and the associated PublishingFile
is returned. No other computations are performed at this point, so

there are no vulnerabilities present.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.9.3 Question 056 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

“When a user goes to their bookmarks page, the BookmarkGuide.handleGrab(...) method is called, which is responsible for building the HTML of the "/bookmarks" endpoint. […]
The method calls inside the map are all very short methods which are either constant-time (looking up the PublishingFiles in the storage, obtaining attributes, etc) or are not influenceable by the user (PUBLISHING_LINK.replaceTags(...), which is a method on a TemplateEngine that replaces template tags with content). These methods also do not expose any space vulnerabilities.
The remaining method calls in the handleGrab() method are straightforward and do not appear to reveal any space or time side-channel vulnerabilities.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.10 Litmedia 2

A.7.2.2.10.1 Question 001 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

“The user's top favorites are only retrieved when the user loads their home page, via a call to MediaLibraryImpl.fetchTopFavorites(...). This method iterates over the existing favorites, sorts them, and takes so many off the top. Since the favorites list is always sorted after every update by a call to updateFavorites(...), the sort in fetchTopFavorites(...) will run in linear time (since it's an implementation of TimSort). This is not exploitable, and so this avenue does not pose a vector for attack.
Users' preferences are also consulted when they select an article to

read. At this point, the article's visit counter (FavoriteNote.tally) is incremented. When this happens, the FavoriteNodes are iterated over sequentially until the node corresponding to the selected article is found, at which point that node's tally is updated and the method returns.

Since these are the only two places where a specific user's favorites are iterated over, there is no vector for attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.10.2 Question 050 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"The user's list of FavoriteNodes are only updated when an article is selected to be read. This means that, in the challenge, the nodes will already have been sorted. When the user clicks on the target literature, its tally will be updated, and the list will be sorted again. This is the only possible vector for attack.

As the sort is unstable, the target literature might be indistinguishable from another literature with the same tally count. Additionally, it would be impossible to deduce how far the target literature's corresponding FavoriteNode moved without first knowing the full ordering of the FavoriteNodes prior to the tally being updated.

Therefore, we conclude that there is no vulnerability present."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.10.3 Question 058 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

"During the session, there will only be one time that the user's bookmarks are retrieved, which is when the user makes the request to load their bookmarks page. […] However, the user's bookmark list is static from this point, so iteration over the returned list gives no side channel

vulnerabilities. […] The remaining relevant methods called in the handleGrab(...) implementation are all constant-time (retrieving a field from a bookmarked media file) or irrelevant (does not concern any specific bookmark). Therefore, there can be no side channel vulnerability to expose a user's bookmarks list."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.11 Phonemaster

A.7.2.2.11.1 Question 003 (SC Time, Intended Vulnerable, Answered: No)

Response:

"When a target User submits a valid request, the checkLogin method validates the username and password. If it's already present in user ArrayList, it returns the User object. Else it adds the object to the list (upon checking the total number of users is less than 151 and password length is greater than or equal to 10) and returns the object. In either scenario, the returned user object isn't used beyond this point. Hence, there is no way to capture the username details from the request submitted by the target User.

The simulate_users script populates the request body(payload) viz username, password, command in a random way. And the validation is not done for username. This makes impossible for the attacker to find the exact username from a target user's request."

Evaluation:

Incorrect; They failed to identify the intended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.12 Powerstate

A.7.2.2.12.1 Question 031 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"There is no vulnerability while parsing the file. Each row is tokenized with a limited number of tokens per row depending on which table its

reading and only reads each file line once. […] While verifying, the only loop that increases space usage and doesn't

have fixed number of iterations will only run at most once per bus. […]

While solving, all objects used are instantiated with a fixed size

based on the number of buses and measurements; with values updated

throughout each iterations given indices.

Therefore, we can confidently state that algorithmic complexity

vulnerability on space is not present in this challenge program."

Evaluation:

Was originally correct but another team discovered an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.12.2 Question 037 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"This attack targets the solver (in class PowerStateSolver), which loops

until the threshold value is low enough. The value being thresholded is

calculated by a sequence of maximum operations in a matrix. We traced

back what influenced this matrix to four core features in the input

tables: Vm and Va in the buses table; and, r and x in the branches

table. For the buses, Vm and Va affect this matrix only if the bus is

of the right type (i.e. not type 3 which is for generators). Therefore

we can set bus 2's Vm to be much larger than the baseMVA and its Va to

be non-zero. For the branches, we need to maximise admittance, which is

inversely proportional to r and x. Thus, r and x must approach 0. This

causes the thresholded value to consistently increase after a certain

number of iterations once it reaches 5E70 it stops.

The above changes to the submission file are enough to cause a delay of

over 25 seconds within the file size restriction, however in our

submission we changed most buses to maximise the delay. Since there are

lengthy validation and parsing requirements changing the sample file

ensures a power network that is properly connected."

Evaluation:

Exploit could not be confirmed

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.13 PSA

A.7.2.2.13.1 Question 010 (AC Time, Intended Vulnerable, Answered: N/A)

Response:

N/A

Evaluation:

N/A

Post-Engagement Analysis Ruling: <Correct/Incorrect>

A.7.2.2.13.2 Question 030 (AC Space (Disk), Intended Vulnerable, Answered: N/A)

Response:

N/A

Evaluation:

N/A

Post-Engagement Analysis Ruling: <Correct/Incorrect>

A.7.2.2.14 Roulette 1

A.7.2.2.14.1 Question 013 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

“The winning number is picked in RouletteWheel.rotate, which decides based on a random value passed into it. The next signal sent from the server is that bets are off (i.e., betting is closed.) As the random number is unknowable and no signals are sent before the "cutoff", no side channel exists.
(Roulettewheel.spin does sleep for a small number of milliseconds based on the inpute, but this is both very precise, and cannot be observed until after it is too late.)”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.14.2 Question 019 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

“The only place in the app that writes a file where it writes the ParticipantsDB. This is a MapDB database of participants and their states. The encoders used in this database are linear and user names are limited to 25 characters so the only way to increase its size is to

insert a large number of entries. Even if every entry took 1kB of disk space, this would require 10^6 participants to exceed the resource usage limit (1kB*10^6=1GB). There is no way to create that many participants within the input budget of 10kB."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.14.3 Question 048 (SC Space, Intended Not Vulnerable, Answered: No)

Response:

"The number of Oracle queries is sufficient to brute force both the type and value, but the amount can vary greatly. An attack would have to work off of some sort of binary search for the bet value. However, the properties of this bet are not widely disseminated through the code, and in particular no queries branch off of them.

A potential route would be to learn the targets bank value before and after they place their bet. The solvency checks can be used to determine if a target bank account is greater or less than a target amount. However, it is not possible to make such checks without access to the target's credentials."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.14.4 Question 054 (AC Time, Intended Not Vulnerable, Answered: No)

Response:

"The program lacks abusable loops or deep recursion. Several calls to thread.sleep are made, but never with parameters that can be stretched.

One avenue of interest is the ArrayBlockingQueue in BetList. If this could be filled, it would cause the "Command" thread (provided by a single thread executor) to block for 120 seconds (by the parameters given in Quantities.)

However, that ArrayBlockingQueue is initialized with Quantties.capacity, which starts at 10,000, and as such is too large to fill within the input budget. Code in DozenWager reduces

Quantities.capacity by a factor of 12, but only after the ArrayBlockingQueue has already been created (so it makes no difference).”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.15 Roulette 2

A.7.2.2.15.1 Question 011 (SC Space/Time, Intended Vulnerable, Answered: Yes)

Response:

“If Game.transmitReminder is set to true, wagerReminderTime is used to determine how long it takes to send out a reminder to post final wagers. This message (measured from the "Wagers On" broadcast) can be used to retrieve the value of wagerReminderTime/lastTime. Game.transmitReminder may be set to true by making a single bet in excess of the attacker's total assets.

lastTime comes up again in game.getTime. That function returns lastTime + a random value, but the random value is very small and varies very little, due to values set in GameParameters (spinMeanTime = 0.01 and spinTimeStanardDeviation = 0.0). This means that lastTime effectively returns spinTime. This return value is the passed (through intermediate values) to RouletteWheel.spin, where it deterministically decides the result.

Putting it all together: The attacker can input a too-large bet request, then measure the time beteween WagersOn and WagerReminder, and use the result to determine lastTime, which can then be used to find the result of the Roulette wheel.”

Evaluation:

Confirmed Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.15.2 Question 033 (SC Space, Intended Vulnerable, Answered: No)

Response:

“The number of Oracle queries is sufficient to brute force both the type

and value, but the amount can vary greatly. An attack would have to work off of some sort of binary search for the bet value. However, the properties of this bet are not widely disseminated through the code, and in particular no queries branch off of them.

A potential route would be to learn the targets bank value before and after they place their bet. The solvency checks can be used to determine if a target bank account is greater or less than a target amount. However, it is not possible to make such checks without access to the target's credentials."

Evaluation:

Confirmed incorrect

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.15.3 Question 046 (AC Space, Intended Not Vulnerable, Answered: No)

Response:

"The only place in the app that writes a file where it writes the PlayersDB. This is a MapDB database of participants and their states. The encoders used in this database are linear and user names are limited to 25 characters so the only way to increase its size is to insert a large number of entries. Even if every entry took 1kB of disk space, this would require 10^6 participants to exceed the resource usage limit (1kB*10^6=1GB). There is no way to create that many participants within the input budget of 10kB."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.16 Securgate

A.7.2.2.16.1 Question 008 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

"All images have the same dimensions, therefore each plate will perform the same amount of requests. However since all plate images have different number of bytes (except for 3.5% of them), it is vulnerable to a side channel space attack. The attacker has access to the plates

folder thus it knows the image sizes. They could observe the payload sizes for packets being sent from the server to the client and figure out which image was being sent. However encryption block sizes are set as a default of 16 bytes. Which makes 29.8% of the images have different sizes, and 30.4% have the same size as one other image.
The images all have dimensions of 1000 by 500, and thus have just over 8000 blocks. If each block consists of 1 TLS session, this leaves enough budget to make a single oracle query. However even with querying one random guess on the oracle, and answering a different plate of the same size it barely gives us theoretical 82.8% success rate. Since there are 50.2% of definitely correct answers, when there are only 2 or less with the same size. We can calculate the probability of succeeding with the other numbers by adding up (fraction of items that have k items with same size)*(probability of picking the right one with the oracle+probability the second one picked was correct). Using the python script attached. We calculated that this gives an 82.8% success rate."

Evaluation:

The script provided only reasons over the set of plates, not the actual observables.

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.16.2 Question 018 (AC Time, Intended Not Vulnerable, Answered: <No)

Response:

"There is no algorithmic complexity vulnerability. get_next_plate doesn't receive any parameters or involve any operations that could be lengthened through algorithmic flaws.
The only possible attack would be a denial-of-service attack due to the large number of synchronized elements in this code, however I believe that is outside of the scope of this problem."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.17 Suspicion 1

A.7.2.2.17.1 Question 020 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

“None of the error log files grow at a substantial enough rate (really, only one line per error, and only one error per request.) The hope would come from MorseCode.unzip, which will attempt to expand whatever is sent. However, this code keeps track of a "sumQuantity" value, and will error out if the result would ever be too large.
In suspicion_2, this can be manipulated by feeding it negative values or values that would make it overflow, causing it to allow too much to be written. The version in suspicion 1, however, has an additional "recordQuantity" call, which checks that the total quantity is never negative and is strictly increasing. This prevents us writing a sufficiently large file.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.17.2 Question 026 (SC Space, Intended Vulnerable, Answered: Yes)

Response:

“A side channel exists due to the byte[] encoding in DistrustCompetition. While this is initially filled with random values, generateAndSendObjectives replaces it with a base 3 representation of the location pointer. This is then used by grabLength, which picks the digit corresponding the current round, which in turn determines the length of the byte array sent (even to the attacker.) Observing the length of the broadcast array yields one (trinary) digit of information. As there are only 26 possible values, after just three rounds it is possible to find the index and guess accordingly.”

Evaluation:

Basically the intended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.2.2.17.3 Question 040 (SC Space/Time, Intended Vulnerable, Answered: No)

Response:

“The password is entered by a user (so, not the result of observable precursors.) It is never sent directly to the spy, and the only non-trivial things done with it are done in the "isPasswordCorrect"

method, which is effectively constant time.”

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.18 Suspicion 2

A.7.2.2.18.1 Question 022 (SC Space/Time, Intended Not Vulnerable, Answered: No)

Response:

“The password is only observed in non-trivial ways in the "DoubtTournament.isPasswordCorrect" method, which is (almost) constant time. A difference of a few cycles exists due to the boolean variable only being modified for non-matching characters, but this is too specific to constitute a valid side channel.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.18.2 Question 044 (AC Space (Disk), Intended Vulnerable, Answered: Yes)

Response:

“MorseCode.expand is called as part of reading input sent to the server; it assumes the data was previously compressed using a run-length encoding scheme, but the attacker can send arbitrary data.
The method stores a temporary file, and writes to the file with repeated runs of bytes (repeated up to an "int" number of times, where the int depends on the subsequent four bytes.) The code attempts to prevent a too-large file by being written, by keeping a running sum of the values entered and checking it against a maximum. This running sum is kept in "aggregateCount"
Nothing prevents this count from being reduced, though. If a negative value is read it will reduce this, but not actually erase part of the file; alternatively, aggregateCount could be made to overflow. Either case will allow a very small input to write a very large output.”

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.2.2.18.3 Question 057 (Null SC Space, Intended Not Vulnerable, Answered: No)

Response:

“The only side channel is the same as was used for the similar question from suspicion_1. That side channel still somewhat exists here, and will encode the location as a three-digit trinary number in a static byte array, whose elements are used to determine the length of additional byte arrays in broadcast messages.

However, this code contains an (apparent) bug where the value of the "Plays" counter is incremented each time getPlays is called. For the other attack to work it is necessary to view each of the first three digits, however, several will be stepped over, depending on what commands are issued by other users. While it is likely possible to retrieve a single digit, that leaves 9 possibilities, and it takes more than 3 operations to get that digit, so the budget is unreachable.”

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.18.4 Question 060 (AC Space (Memory), Unintended Vulnerable, Answered: Yes)

Response:

“MorseCode.expand is called as part of reading input sent to the server; it assumes the data was previously compressed using a run-length encoding scheme, but the attacker can send arbitrary data.

The method stores a temporary file, and writes to the file with repeated runs of bytes (repeated up to an "int" number of times, where the int depends on the subsequent four bytes.) The code attempts to prevent a too-large file by being written, by keeping a running sum of the values entered and checking it against a maximum. This running sum is kept in "aggregateCount"

Nothing prevents this count from being reduced, though. If a negative value is read it will reduce this, but not actually erase part of the file; alternatively, aggregateCount could be made to overflow. Either case will allow a very small input to write a very large output.

After this file is created on disk, Files.readAllBytes() is called on

it, and read into a byte[] array in memory."

Evaluation:

Confirmed unintended vulnerability

Post-Engagement Analysis Ruling: Correct

A.7.2.2.19 ThermoMaster

A.7.2.2.19.1 Question 021 (AC Space (Disk), Intended Not Vulnerable, Answered: No)

Response:

"Note that the `Server.serve()` is a `synchronized` method. Thus only one instance of `Server.serve()` can be running at a time regardless of whether there are multiple simultaneous requests running. This considerably simplifies the analysis.

The only file written by the app is the log file. This is written by a logger thread in a loop. The only place that data is sent to the logger is when `ThermoMaster.log.receiveLogInfo()` is called by `PID.run()`, which in turn in started by `Server.update_controller()`. Before each `PI.run()`, the variable `ThermoMaster.log.newLog` is set to false by `Server.update_controller()`. This will cause the logger thread to reopen and thus overwrite the log file the next time through the logger loop.

Thus the maximum size of the log file is determined by how much log data can be written within one run of `PID.run()`. The loop in `PID.run()` iterates a maximum 32,000,000 times, with one out of every 1000 runs adding 20 bytes to the log. Thus the largest file created would be only 640kB large. This is under the resource usage limit so there is no attack."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.19.2 Question 038 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"Note that the `Server.serve()` is a `synchronized` method. Thus only one instance of `Server.serve()` can be running at a time regardless of whether there are multiple simultaneous requests running. This

considerably simplifies the analysis.
The benign user's setpoint is leaked to the global `Server.setpoint` variable. However, this is reset to `null` at the end of `Server.serve()`. The only leak would be if the user's setpoint affected the run time of the benign user requests. Any attacker requests will not be affected by the benign user's setpoint.
Furthermore tracing the data flow of Server.setpoint shows that it does not affect the runtime of user requests *except* when the runtime is affected by the difference from the user's current temperature. This makes it impossible to isolate the value of the benign user's setpoint."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.19.3 Question 052 (AC Time, Intended Vulnerable, Answered: Yes)

Response:

"The attack is accomplished by the attacker sending the example request (examples/run_example.sh) (with a modified username and password) immediately before the benign request. Since `Server.serve()` is `synchronized`, this forces the benign request to wait to start until after the attacker's request finishes. The example request takes 9 seconds to complete. In addition, the *benign* example request takes 9 seconds to complete before it responds. Thus the total time taken is 18 seconds which is above the resource usage limit. Sending the example request involves 534 bytes of data which is within the input budget.
This attack does rely on the attacker being table to figure out a way to send the attack within the three (18 minus 15) second window before the benign request. This can be accomplished by withholding the last byte of the attack request. When the first bytes of the TCP/SSL handshake of the benign request are observed by the attacker, the attacker can send the last byte of the attack request. Since the handshake has already been completed by the attacker, the attacker's request will be processed first."

Evaluation:

Confirmed this exploit

Post-Engagement Analysis Ruling: Correct

A.7.2.2.20 Wordshare

A.7.2.2.20.1 Question 002 (SC Space, Unintended Vulnerable, Answered: No)

Response:

"When a document with secret word is uploaded, client checks whether the secret length is less than or equal to 16 bytes. If not, it includes space character (in bytes) to make it as 16. This is then encrypted using AES(with secret key) and prepended with 16 byte initialization vector. The output html file will have the 32 byte encrypted base64 encoded secret word, irrespective of its input size of the secret. Further, the response size sent by the server is also irrespective of the secret word we enter in the input document.
Hence, we conclude there isn't any side channel vulnerability in space from which an attacker with user key would be able to discover the plaintext of any secret word."

Evaluation:

Was originally correct but another team discovered an unintended vulnerability.

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.20.2 Question 005 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

"[…] The Base64 encoded string of the encrypted text is replaced in the div tag in the client side and sent to server. The other contents of the input are stored without any modifications in its size.
We could also observe that the secret word and input file content are stored in the trie structure. For each alias file name a new WordShareDoc object is created. But the output file is written only once.
Hence, given the input budget of 250KB it is not possible to generate an output file with size 35MB."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.20.3 Question 047 (SC Time, Unintended Non-Vulnerable, Answered: Yes)

Response:

"Step 1) In Search operation, Start with a string with one character. Try each of 26 letters and save the response and the time taken to produce response. If the letter is present only in the secret document, we might get occurrence as 0. If the occurrence is 0, and time is max that is the perfect candidate. If there are no results with zero occurrences, find the time taken to occurences found ratio of all the letters. Pick the letter with max ratio. It is more likely that this letter is part of a secret word sequence.
Step 2) Store the time it took to find letter 1. Now attacker sends strings of length 2 with the first letter found from step 1. Again start trying out all alphabets for the second letter. Ignore all the letters that are taking lesser time than time taken to find letter 1. They are more likely not our targets. Store the time and occurrence ratio for all the other letters and repeat step 1. i.e pick the max ratio letter and proceed. Repeat steps 1 and 2.
Step 3) There will be a point where time taken for current letter is same as previous letter. This means we are on the right track in finding the secret. Continue on (will be getting same times for all remaining letters in the secret since the trie has to parse the whole matched word to find the file name). Continue until the time taken is same for subsequent letters. Stop at the point where it changes. This means we have reached the end of the secret. As we try to find the end of the secret, if we are getting one occurrence as the search result, then it means there is another non-secret word that has the same letter sequence. In that case, we delete that file since we know the file name and repeat the whole process with the found sequence of words so far to find the secret word."

Evaluation:

Utah has found an instance where an attack works for a limited set of the possible secrets, but not all and not in the worst case. This challenge was intended vulnerable but was found to be non-vulnerable for the worst-case secret.

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.21 YapMaster

A.7.2.2.21.1 Question 015 (AC Space (Disk), Intended Vulnerable, Answered: No)

Response:

"If we are to increase the size of only one file then that will mostly be the .map files generated for the aliases. The .map files for alias store the message information posted. If any modification are made to the message then this modification is also stored in the .map files. The sharing option does not increase the size of the .map file. The SvrLogic has access to these .map files. And even then only to add a yap, modify a yap and get a yap. No other flow has access to the .map files. The yaps while adding and modifying are checked for length and allowed characters and it is placed in cache and written to the .map files. Hence given the budget of the attack and the limit that we have to reach exploit is not likely."

Evaluation:

Incorrect; they failed to find the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

A.7.2.2.21.2 Question 017 (SC Time, Intended Not Vulnerable, Answered: No)

Response:

"The only promising flow for this attack was the cache flow in the app. The yaps are stored in Cache which we can retrive if we know the position. However the yaps in the cache is are encrypted using a key which is generated using among other things the user seed. The other flow relies on the the user seed to retrieve data from the .map files so given the scenario we cannot see an exploit."

Evaluation:

Correct

Post-Engagement Analysis Ruling: Correct

A.7.2.2.21.3 Question 042 (SC Space, Intended Vulnerable, Answered: No)

Response:

"The only promising flow for this attack was the cache flow in the app. The yaps are stored in Cache which we can retrive if we know the position. However the yaps in the cache is are encrypted using a key which is generated using among other things the user seed. The other flow relies on the the user seed to retrieve data from the .map files so given the scenario we cannot see an exploit."

Evaluation:

Incorrect; they failed to identify the intended vulnerability

Post-Engagement Analysis Ruling: Incorrect

Appendix B: Study of Side Channel in Emissions

List of Tables

List of Figures

In addition to experiments described above, we performed a study and analysis to understand the bounds on using RF timing side channels to infer information about the underlying software being run on a system. We considered several use cases in which this side channel would be pertinent:

a) One possesses a copy of the program that is running and is using the emanations to identify what execution paths the program is executing.
b) One has an old version of the program that is running and is using the emanations to both identify what dynamic traces the program is executing and what portions of the program have been modified.
c) One knows approximately what the program is doing and is using the emanations to identify specifically which algorithm is being applied (e.g., one is aware that the program uses Fourier transforms and wants to identify which Fourier transform (or which well-known implementation) is in use).

It is noted that use case (a) is at the heart of Alam, et al's *One&Done: A Single-Decryption EM-Based Attack on OpenSSL's Constant-Time Blinded RSA* (27th USENIX Security Symposium), which describes using the radio eminations side channel of a CPU to capture an RSA private key.

The problem is exacerbated by a number of factors:

- One does not map an RF signal to an individual instruction, but rather to a fixed-length sequence of instructions. And it is not mapped to a specific sequence, but rather is mapped as a probability distribution over the set of possible instruction sequences, where that set is constrained by what one knows about the program being executed. Thus, under optimal conditions, there is significant uncertainty regarding which code block is currently being executed.
- One cannot assume that the sensing begins when the program begins. Rather, one assumes that the program has been running for some period of time when the sensing commences, so at time τ_0 one does not know where one is in the execution.
- One cannot assume that the sensing interval is synchronized with the program, and so one cannot define fixed intervals in the code at which the "code blocks" begin and end.
- Finally, we are assuming a lossy-signal environment in which there will be periods of time in which no signal is being received.

The purpose of this analysis task is to is to evaluate the bounds of this timing side channel. E.g.:

- How much a-priori knowledge one must have of the program that is leaking information.
- How accurate and precise must the signal be in identifying code blocks.
    - Similarly, how small a sequence of instructions is mapped to a signal.
- How frequently must one observe the signal? In other words, for how many code blocks can one not receive a sensor input and still be able to classify the computation.
- classify the computation.

The three forms of the problem enumerated above (a, b, c) were listed in order from easiest to most difficult; we focused on (a), performing a *practical* analysis (we will examine representative examples) and then use the lessons learned to make theoretical claims/hypotheses.

The Analysis

We looked at the papers and briefings of one of the current LADS performers to assess *their* analysis of the R/F emissions leakage. Their claim of being able to correlate R/F emissions to the code that is being executed appears to be well founded. However, they are not doing a root cause analysis of the source of the emissions. So, it is not clear whether the emissions are generated by the code itself or by CPU activity that is correlated with the instructions being executed. We also determined that the LADS performers are not treating this as a tracking (or multi-hypothesis tracking) problem. Or even as a hidden Markov model. The approaches appear to be very ad hoc.

We developed an exemplar architecture for treating problem (a) as a multi-hypothesis tracking problem. An aspect of that is to develop a control flow graph (CFG), so as to develop prior probabilities. (If one is at point X in the program with probability p_1, then in the future one will be at point Y of the program with probability p_2.) We examined the tools available for building CFGs statically (they work well inside of a shared object [*.so or executable], but do not work well across shared objects) and dynamically (from trace). Each approach has advantages and disadvantages: static analysis does not reveal the probability that a given edge will be traversed; dynamic analysis is laborious, and one must be able to provide varied and "typical" inputs to the program.

We developed a document model to represent the code blocks in a program, which allows one to leverage the science of information retrieval to analyze the code. Individual instructions (opcodes) were treated as characters; basic blocks as words; and functions as documents. Thus, the program is a collection of documents, and one is using information retrieval to characterize the probability that an observed word originated from a given document.

A discovery is that the distributions of the in-degree and out-degree of nodes in the CFG follow a power-law distribution; a few nodes have a large number of predecessors / successors in the directed graph, while a large number of nodes have few predecessors / successors. (See Figure 87.)

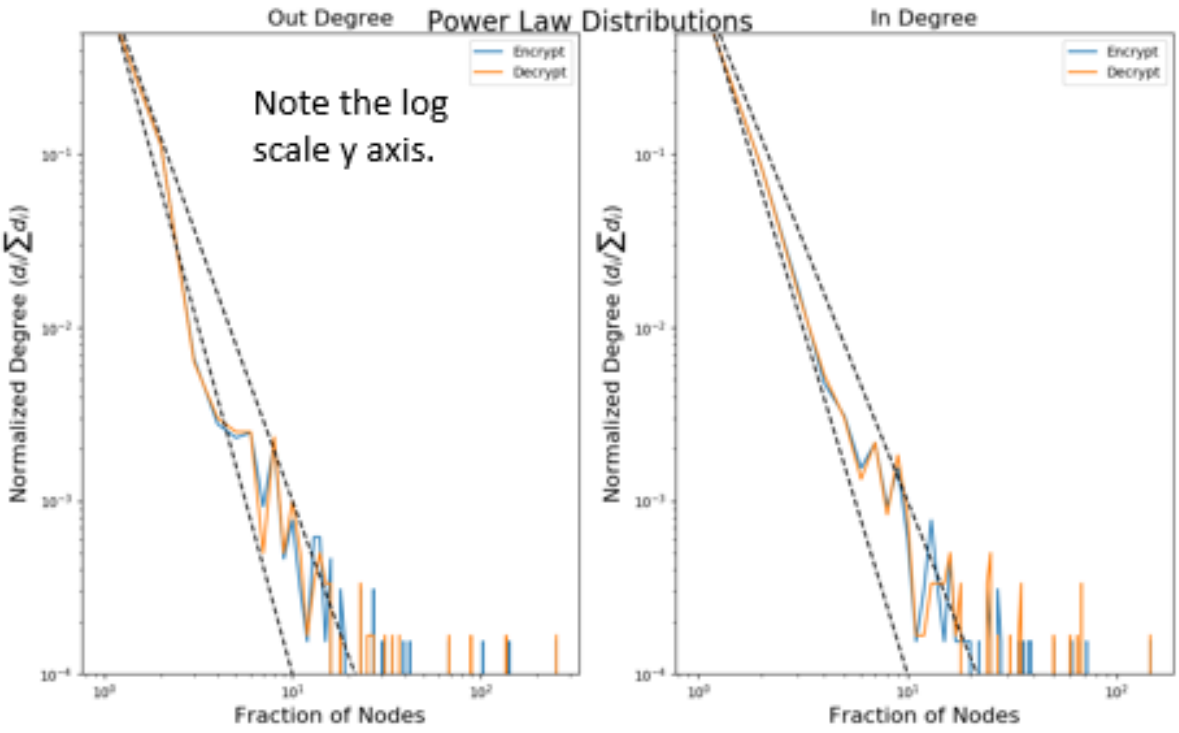


Figure B-1: Power Law Distribution of In- and Out-Degrees in openssl

Another discovery is that the number of unique words in each function (document) also follows a power law. Recall that a word is a sequence of instructions. A *unique* word is a sequence of instructions that appears in a single location in the program. I.e., given all of the libraries that the program traverses, that sequence of instructions is located at a single address. If one can use R/F emissions to identify a sequence of instructions, then a unique word allows to observer to identify exactly where the program is with regards to its control flow graph.

As shown in Figure B-1, a very few functions have a large number of unique words in them, while many functions have vew few or no unique words.

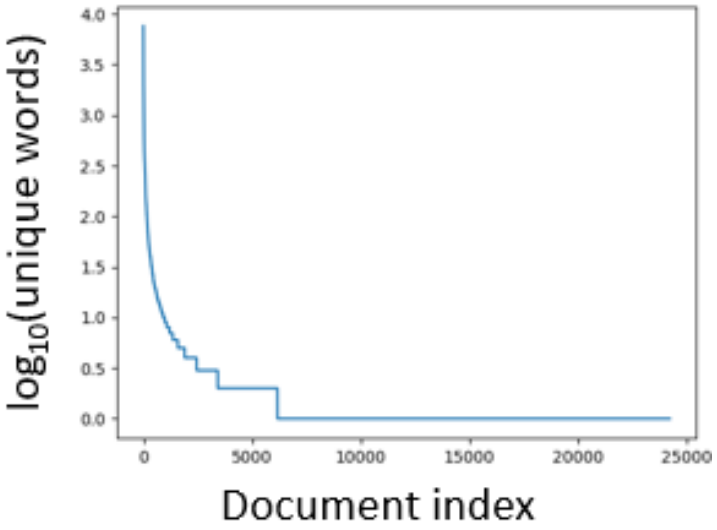


Figure B-2: Number of Unique Words in Each Function

An interesting aspect of unique words is that, when they are located in a library function, they can tell the listener not only which function is being executed and which library is being executed but also which version of that library. This is particularly useful in the case of libc, and the operating system version is often closely tied with a specific libc version. This is illustrated in Figure B-2., which maps the recent Raspberry Pi kernel images with their associated libc version. Thus, if one can use R/F emissions to identify code sequences, one can identify the libc version that is executing, which in turn reveals which kernel is installed.

Table B-1: Mapping the Raspberry Pi kernel versions to their libc version

Linux Image Version	Libc Version
"wheezy": kernel 3.18.7+ (2015-02-16)	libc-2.13
"jessie": kernel 4.1.19-v7+ (2016-03-18)	libc-2.19
"stretch": kernel 4.14.71-v7+ (2018-10-09)	libc-2.24
"buster": kernel 4.19.66-v7+ (2019-07-10)	libc-2.28

Interestingly, libc has a higher portion of unique words in it than do many other libraries. We do not as yet have a hypothesis to explain this. In Figure B-3, we show the number of unique and non-unique words across all of the openssl traces we collected, assigned to the libraries in which they appeared. The ARM linux loader, libc and libcrypto have exceptionally high fraction of their words (that appear in the traces) be unique words. In contract, libssl, libdl, libpthread and libarmmem have very low percentages of unique code words.

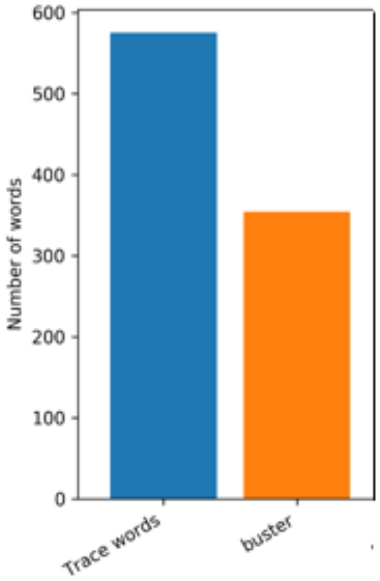


Figure B-3: Comparing the Words Unique to libc to th Words Unique to version 2.28 of libc

Our traces were collected on a *buster* kernel—they all used version 2.28 of libc. Figure B-4 shows the number of libc-unique words that appeared in the openssl traces (there were 575 of them) and the number of those words that are unique to libc 2.28 / buster (i.e., would identify not only libc execution but also identify the operating system).

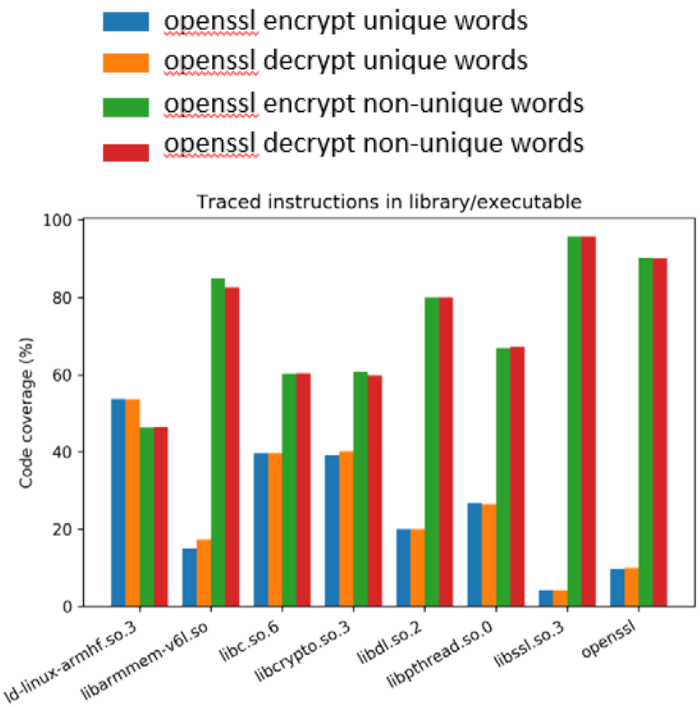


Figure B-4: The Number of Unique Words Encountered in openssl

Finally, we examined *Perf* (also called *perf_events*) as tool to support R/F side channel analysis. Perf is a tool for performance analysis on Linux platforms. It monitors *events*, both tracking timestamps for those events and capturing sequences of events. These can be *soft* or *hard* events—events that are software generated or are supported/generated by the hardware. Due to its prevalence and sophistication, the x86 family of processors (both Intel and AMD) are very well supported by perf. ARM CPUs less so. This is illustrated via the command "perf list" which lists the performance counters and events that are available on the platform. If one runs the output through "wc -l" (counting the lines), on ARM one gets 66 results, but on an Intel xeon processor one gets 2,035 individual trackable items.

That said, one does not require a large number of trackable items for perf to support side channel R/F analysis—it only needs to support the right trackable event. We identified *uprobe* as the event of interest. *uprobe* is a soft event; it is a technique for marking an individual instruction as a traceable event. One can, from the command line, tell perf which instruction of any given program or library to track, and then see timestamps (with the granularity that the platform's clock supports) when that/those instruction(s) was/were executed. Using this feature, one can identify specific points in a program to monitor and use static analysis to codify the code that was exected between those points, resulting in a relatively complete picture of the execution. This can be enhanced by perf's ability to track cache misses, page faults, process swaps, system calls, and pipeline stalls—all events that impact inter-instruction timing.

Bibliography

"Thinly Provisioned Logical Volumes," Red Hat, URL: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Logical_Volume_Manager_Administration/thinprovisioned_volumes.html

"LVM Logical Volumes," Red Hat, URL: https://access.redhat.com/documentation/en-US/Red_Hat_Enterprise_Linux/6/html/Logical_Volume_Manager_Administration/lv_overview.html#linear_volumes

List of Symbols, Abbreviations, and Acronyms

AAD	Answers per Analyst Day
ABC	a counting methodology
AC Teams	Adversarial Challenge Teams
AC Time	Algorithmic Complexity Time
ACK	Acknowldge
ACT	Algorithmic Complexity Time
ACV	Algorithmic Complexity Vulnerability
ACV-S	Algorithmic Complexity Vulnerability-Space
ACV-T	Algorithmic Complexity Vulnerability-Time
AEL	Adjusted Expenditure Limit
AES	Advanced Encryption Standard
AFL	American Fuzzy Lop
AGL. COMP.	Algorithmic Complexity
AI	Artificical Intelligence
AMD	Advanced Micro Devices, Inc
ANL	Article Name Length
API	Application Programming Interface
ARM	Arm Holdings
ASCII	American Standard Code for Infomration Interchange
BAA	Broad Agency Announcement
BBN	Raytheon/BBN Technologies
BCC	Blind Carbon Copy
BIOS	Basic Input/Output System
BMP	Bitmap image file
BPM	Beats Per Minute
BSSIDs	Basic Service Set Identifiers
CAGE	Complexity Analysis-based Guaranteed Execution
Case DB	Case Database
CBC	Cipher Block Chaining
CBC	Cypher Block Chaining
CC	Carbon Copy
CD	Certified Deposit
CentOS	Community Enterprise Operating System
CFG	Control Flow Graph
CFR	Class File Reader
CHAID	Chi-Squared Automatic Interation Detector
CLI	Command Line Interface
CMA-ES	Covariance Matrix Adaptation Evolution Strategy
CP	Cyberpoint
CPU	Central Processing Unit

CRA	Compositional Recurrance Analysis
CRC	Cyclic Redundancy Check
CRIME	Compression Ratio Info-leak Made Easy
CRSF	Cross Site Request Forgery
CRT	Chinese Remainder Theorem
CSRF	Cross Site Request Forgery
CSV	Comma Separated Values
CTR	Counter
CVE	Common Vulnerabilities and Exposures
CWE	Common Weakness Enumeration
DARPA	Defense Advanced Research Projects Agency
DB	Database
DES	Data Encryption Standard
DFS	Depth First Search
DH	Diffie-Hellman
DHCP	Dynamic Host Configuration Protocol
DHE	Diffie-Hellman Key Exchange
DOM	Document Object Model
DoS or DOS	Denial of Service
ECDSA	Elliptical Curve Digital Signature Algorithm
ECJ	Java Evolutionary Computation Toolkit
EFG	Event Control Flow Graph
EFI	Extensible Firmware Interface
EL Teams	Experimentation Lead Teams
FIFO	First In First Out
GB	Gigabyte
GCM	Galois/Counter Mode
GPX	GPS Exchange Format
GRAY	a image filetype
GRUB2	Grand Unified Bootloader 2
GUI	Graphical User Interface
HIV	Human Immunodeficiency Virus
HMAC	Hash-based Method Authentication Code
HSP	Hash of Salted Password
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
HTTPS	Hypertext Transfer Protocol Secure
I/O	Input/Output
ICRA	Interprocedural CRA
ID	Identifier
IEEE	Institute of Electrical and Electronics Engineers
IM	Inventory Manager
IP address	Internet Protocol Address

IPCG	Interprocedural Call Graph
IPv4	Internet Protocol Version 4
ISU	Iowa State University
IV	Initialization Vector
JAD	Java Decompiler
JAR	Java Archive
JD-GUI	Java Decompiler-Graphical User Interface
JDK	Java Development Kit
JIT	Just-in-Time compilation
JPEG	Joint Photographic Experts Group
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
KB	Kilobyte
LADS	Leveraging the Analgo Domain for Security
LattE	Lattice Point Enumeration
LCG	Loop Call Graph
LED	Law Enforcement Database
LFSR	Linear Feedback Shift Register
LPCG	Loop Termination Projected Control Graph
LVM	Logical Volume Manager
LVM2	Logical Volume Manager 2
LZW	Lempel-Ziv-Welch
MAC	Message Authentication Code
MB	Megabyte
MCS	Main-Channel Size
MD5-hash	Message Direct Hashing Algorithm 5
MIME	Multipurpose Internet Mail Extensions
MST	Minumum Spanning Tree
MTU	Maximum Transmission Unit
NA	Not Applicable
NAT	Network Address Translation
NB	Nota Bene, “Note Well”
NE	Northeastern University
NEU	Northeastern University
NFA	Non-Deterministic Finite Automata
NIC	Network Interface Card
Non-SC	Non Side Channel
NP	Non-Polynomial
NP-hard	Non-Polynomial Time Hard
NRE	Non-Required Expenditure
NUC	Next Unit of Computing
NYC	New York City
OS	Operating System

PC	Personal Computer
PCG	Program Control Graph or Projected Control Flow Graph
PDU	Protocol Data Unit
PEL	Project Expenditure Limit
PF	Project File
PI	Principal Investigator
PID	Proportional-Integral-Derivative
PM	Program Manager
PNG	Portable Network Graphics
PoC or POC	Proof of Concept
POI	Point of Interest
POV	Proof of Vulnerablility
PRNG	Pseudorandom Number Generator
PS	Post Script
QCHL	Quantitative Cartesian Hoare Logic
R&D	Research & Development
RAM	Random Access Memory
RF	Radio Frequency
RGB	Red Green Blue
RGBA	Red Green Blue Alpha
RLE	Run-length Encoding
RNG	Random Number Generator
ROC	Reciever Operating Characteristic
RPC	Remote Procedure Call
RSA	Rivest-Shamir-Adleman Encryption
RSS	Rich Site Summary
SAX	Simple API for XML
SC	Side Channel
SCL	Stacked Complementary Losses
SCL-S	Side Channel Leak-Space
SCS	Side-Channel Size
SC-S	Side Channel-Space
SDL	Simple Direct Media Layer
SEDA	Staged Event-Driven Architecture
SHA	Secure Hash Algorithm
SHA1 or SHA-1	Secure Hash Algorithm 1
SPF	Symbolic Pathfinder
SPF-SC	Symbolic Pathfinder-Side Channel
SPF-WCA	Symbolic Pathfinder-Worst Case Analysis
SQL	Structured Query Language
SSD	Solid State Drive
SSL	Secure Sockets Layer

STAC	Space/Time Analysis for Cybersecurity
STD	Standard
STDIN	Standard Input
STDIO	Standard Input/Output
STDOUT	Standard Output
TA1	Technical Area 1
TA3	Technical Area 3
Tawa FS	Tawa File System
TCP	Transmission Control Protocol
TD	Table Data
TLS	Transport Layer Security
TNR	True Negative Rate
TPR	True Positive Rate
TSP	Travelling Salesman Problem
UC Boulder	University of Colorado Boulder
UCSB	University of California: Santa Barbara
UDP	User Datagram Protocol
UEFI	Unified Extensible Firmware Interface
UI	User-Interface
UMD	University of Maryland
URI	Uniform Resource Identifier
URL	Uniform Resource Locator
USB	Universal Serial Bus
Utah	University of Utah
UTF	Unicode Transformation Format
UUID	Universally Unique Identifier
WALA	Watson Libraries for Analysis
WAV	Waveform, a filetype
XCSG	Extensible Common Software Graph
XFS	Extents File System
XML	Extensible Markup Language
ZIM	a filetype
ZIP	a filetype
ZLIB	a software library